I.E.S. Francisco Romero Vargas
(Jerez de la Frontera)

Administración de Sistemas Informáticos en Red

Curso: 2023/2024

PROYECTO INTEGRADO LITTLE ALCHEMY EN AWS



Realizado por:
Héctor Lasanta Martín
Tutor asignado:
Aurelio Barreda Guerrero

Tabla de contenido

Introducción

En este documento se detalla toda la información necesaria para comprender el concepto, la implementación y el funcionamiento del proyecto de fin de grado del autor. La idea consiste en el desarrollo de una aplicación web y la gestión de todos los recursos necesarios para permitir su correcto funcionamiento y administración.

La aplicación es un "juego" accesible desde el navegador inspirado en el proyecto *Little Alchemy*, desarrollado por Recloak Games y lanzado en diciembre de 2010, en el que los jugadores deben combinar elementos para generar otros nuevos, hasta descubrir las 580 combinaciones disponibles.

Figura 1

Logo de Little Alchemy

Little Alchemy

Nota. Logo del juego Little Alchemy. Realizado por Recloak Games, 2010

Como se detalla en los apartados siguientes, el proyecto ofrece una versión alternativa del juego original que utiliza una interfaz de programación de aplicaciones (API) para generar combinaciones infinitas de elementos mediante el uso de un modelo de lenguaje grande (LLM).

Finalidad y objetivos

El proyecto, al tratarse del desarrollo y la implantación completa de un producto; y no un servicio ideado para mejorar la situación de una empresa, tiene como objetivo conseguir que dicho producto, en este caso la aplicación web, sea implementado de manera eficiente, de fácil expansión y asegurando la alta disponibilidad del mismo.

Para ponerlo en perspectiva, imagine que una empresa quiere desarrollar la página web para un juego y subirla a internet para que los usuarios puedan acceder sin problemas al mismo. Para ello, contrata a dos equipos: un primer equipo que se encargará de desarrollar la aplicación, y un segundo equipo que tendrá como objetivo la administración y gestión de todos los recursos necesarios para que el primer equipo pueda lanzar la aplicación una vez desarrollada (servidores, copias de seguridad, configuración de red, reglas de *firewall*, control de permisos, obtención y implantación de certificados y dominios, etc.).

Teniendo ya en mente esa situación concreta, la finalidad del proyecto es ponerse en el papel de ambos equipos a la vez y conseguir alcanzar todos los objetivos que cada uno debe cumplir para conseguir que la aplicación web sea desarrollada, implementada y servida al usuario de manera correcta.

Medios necesarios

Como la intención tras el proyecto es conseguir que se asemeje lo más posible al desarrollo de una aplicación real, queda descartada la opción de trabajar de manera local con ordenadores físicos o máquinas virtuales en una subred privada dentro de un portátil, ya que sería bastante complicado y costoso permitir que la aplicación fuera accesible desde cualquier lugar sin sufrir las consecuencias de un tráfico de red potencialmente elevado.

Es por esto que el proyecto no utilizará equipos físicos para albergar los servidores, y en su lugar, se trabajará con máquinas virtuales creadas en Amazon Web Services (AWS). Como consecuencia, la lista de medios y recursos necesarios para la realización del proyecto, queda reducida a:

- Un equipo con conexión a internet para acceder a la consola web de AWS y conectarme por SSH a las máquinas virtuales.
- Una cuenta de AWS personal o proporcionada por el instituto con acceso a la capa gratuita de AWS.
- Una cuenta de pago que permita acceso ilimitado a una API que envíe a la IA las consultas necesarias o alternativamente el modelo local de Llama2 ofrecido gratuitamente por Meta para descargarlo en una máquina virtual.
- Un dominio personal o en su defecto un subdominio gratuito que permita acceder a la página web sin necesidad de introducir la dirección IP.
- Acceso a una biblioteca gratuita de imágenes que permitan implementar todas las funciones necesarias para el correcto funcionamiento de la aplicación.

Justificación

El autor tiene constancia de que podría dar la impresión de que el proyecto es uno más relacionado con el desarrollo web que con la administración de sistemas, y por tanto, inadecuado para el módulo cuya calificación depende en gran medida del proyecto y de su relación con lo estudiado a lo largo de los últimos dos años. Se comprende la situación en la que se encuentra el proyecto, y por lo tanto se considera necesario justificar con ejemplos concretos la elección de este proyecto.

A continuación se ofrece una lista con cada uno de los módulos cursados y todos los elementos del proyecto que guardan una estrecha relación con dichos módulos:

- **Implantación de sistemas operativos:** Para poder realizar el proyecto ha sido extremadamente importante el dominio de los comandos de Linux, así como su estructura de ficheros y gestión de permisos para los usuarios. Las máquinas virtuales utilizan debian 12, así que estos aspectos han estado presentes en todos y cada uno de los días del desarrollo, y de no haberse aplicado, jamás se habría conseguido realizar este proyecto.

- **Planificación y administración de redes:** Todos los conceptos sobre redes aprendidos en este módulo han sido de gran ayuda a la hora de planear la red virtual privada en la nube (VPC). Además, la mayor parte de los conocimientos sobre AWS del autor se originan en este módulo.

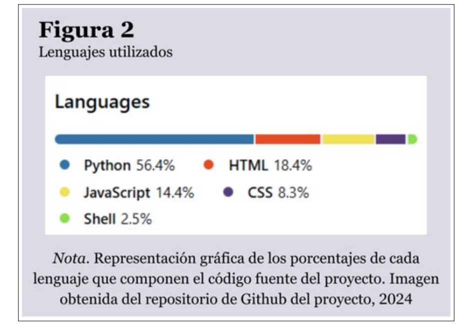
- **Fundamentos de *hardware*:** Aunque no ha sido necesario hacer uso de ninguna máquina local, este es el módulo en el que se aprendió a utilizar *git*, que ha sido una de las herramientas que más se han utilizado para la parte de desarrollo del proyecto. Poner en práctica estos conocimientos ha demostrado que lo aprendido en esta asignatura resultaba más importante de lo que se imaginaba el autor en un primer lugar. Si esto no fuera suficiente, fue en esta asignatura en la que el autor

utilizó su primer *framework* para Python, Flask, que aunque no terminó de resultar convincente, facilitó la transición a Django, el *framework* utilizado en este proyecto.

- **Lenguajes de marcas y sistemas de gestión de información:** Absolutamente todo lo aprendido en este módulo es de vital importancia para un proyecto como el que se propone. Saber trabajar con HMTL, CSS y JavaScript es algo fundamental para el desarrollo de una aplicación web básica como la planteada, como queda demostrado por el hecho de que el 40% del código fuente del proyecto una vez terminado está compuesto por estos tres lenguajes.

  Por si eso no fuera suficiente, en este módulo también se trabajó con *markdown, bootstrap y Github*, siendo los tres elementos que se utilizan en el proyecto en mayor o menor medida.

  Figura 2
  Lenguajes utilizados

  Nota. Representación gráfica de los porcentajes de cada lenguaje que componen el código fuente del proyecto. Imagen obtenida del repositorio de Github del proyecto, 2024

  El autor considera que este ha sido posiblemente el módulo más importante para la realización del proyecto, ya que toda la página web, los estilos, el diseño responsivo, los elementos dinámicos, etc. que son las piedras angulares del proyecto, se han creado en base a lo aquí aprendido.

- **Gestión de bases de datos:** A pesar de que Django funciona mediante un sistema de modelos que en un principio es totalmente diferente a los conocimientos de MariaDB aprendidos en este módulo, los conceptos generales sobre bases de datos relacionales así como el nivel de abstracción requerido para diseñarlas fueron de vital importancia a la hora de implementar todo el sistema de bases de datos del proyecto, como quedará explicado en un apartado futuro de este documento.

- **Administración de sistemas operativos:** En este módulo se trató de nuevo el uso de diversos sistemas operativos basados en Linux, lo que ayudó a reforzar los conocimientos que ya mencionados antes al hablar de ISO. Lo que este módulo aportó y que ha sido de gran utilidad, fue todo lo aprendido en relación a la automatización de *scripts* y procesos, que ha jugado un papel fundamental en la configuración del servidor de copias de seguridad.

- **Servicios de red e internet:** Gracias a este módulo el autor aprendió todo lo necesario para configurar un servidor Nginx para que funcionase como lo necesitaba. Nuevamente esto se tratará con más detalle en un apartado futuro. El proceso de creación / obtención de certificados y subdominios, así como el uso de HTTPS también han sido importantes para el proyecto. A todo esto se suma lo aprendido sobre el sistema de nombres de dominio (DNS), que ha sido necesario configurar para que al introducir el nombre de dominio correcto el usuario sea remitido a la dirección IP del servidor.

- **Implantación de aplicaciones web:** A pesar de que no se ha utilizado PHP, los conceptos generales de programación, especialmente sobre sesiones y *cookies*, sí que se han puesto en práctica, resultando bastante útiles.

- **Administración de sistemas gestores de bases de datos:** Ya se mencionó que el 40% del código era una combinación de HTML, CSS y JavaScript; pues el 60% restante es Python, lenguaje de programación que se aprendió a usar en este módulo durante el primer trimestre. También se trabajó de nuevo con distintos *frameworks* de Python, lo que creo que justifica el uso de uno para el proyecto.

Figura 3
Frameworks de Python conocidos



Nota. Logos de los *frameworks* de Python conocidos por el autor. De izquierda a derecha: Flask, Tkinter, Django. Las imágenes pertenecen a los proyectos homónimos.

- **Seguridad y alta disponibilidad:** De este módulo ha sido imprescindible aplicar lo aprendido sobre reglas de seguridad, ya que el "*firewall*" de AWS debe configurarse de una manera muy específica para permitir el funcionamiento de todos los elementos del proyecto a la vez que se garantiza la máxima seguridad posible de los mismos.

Desde el punto de vista del autor, los anteriores apartados demuestran con una claridad más que suficiente que no solo es este un proyecto relacionado con lo estudiado, sino que se trata de uno que intenta aplicar y expandir los conocimientos adquiridos en todos y cada uno de los módulos cursados.

Planificación

Este proyecto comenzó su desarrollo el lunes 18 de marzo de 2024. A continuación se detalla el progreso que se esperaba conseguir cada semana hasta la fecha de entrega del proyecto, siendo esta el viernes 7 de junio de 2024.

Semana 1: Esta semana será dedicada principalmente a la búsqueda de información y a la realización de pruebas de concepto para poder ir orientando el proyecto correctamente.

Semana 2: Durante la segunda semana se pretende conseguir la estructura básica de la aplicación web funcionando en Django con funcionalidades limitadas y estilos CSS mínimamente aceptables. Esto incluye la conexión con una base de datos local (temporal) para poder hacer pruebas

Semana 3: Sería ideal conseguir implementar la API de Llama2 durante esta semana para garantizar que funciona correctamente y realizar un cambio de planes en caso de que fuera necesario. También se espera que esta semana se empiece a probar la mecánica de combinación de elementos, a poder ser con un *pop-up* personalizado.

Semana 4: Suponiendo que la API sea implementada correctamente, la cuarta semana sería dedicada a conseguir que la IA combinara los elementos correctamente y registrara

en la base de datos las combinaciones. Si sobrase tiempo tampoco estaría mal mejorar los estilos un poco.

Semana 5: Una vez implementadas las funciones básicas, lo siguiente sería dedicarle un tiempo a depurar el código y a controlar errores para garantizar una base robusta sobre la que construir los detalles finales a implementar durante las próximas semanas. El tiempo extra, de haberlo, se dedicará a retocar archivos existentes.

Semana 6: Esta semana se implementará el sistema de registro e inicio de sesión para la aplicación de Django. Si sobrase tiempo, que es probable, estaría bien dedicarlo a la implementación del sistema de estadísticas.

Semana 7: Durante la séptima semana se trabajaría en todo el contenido dependiente de cookies. Esto incluye mensajes de bienvenida y un tutorial que explicaría en detalle como funciona la aplicación y cuáles son sus funciones. Si fuera posible, sería interesante añadir algunas animaciones sencillas con CSS para mejorar la presentación.

Semana 8: Ya que se aproxima la fecha de finalización del proyecto, esta semana sería un buen momento para migrar la base de datos a una instancia RDS de AWS y para configurar el servidor Nginx (incluyendo HTTPS), dejando así un margen razonable en caso de que algo no funcionara como se espera.

Semana 9: El proyecto base ya debería estar más que terminado, así que este tiempo se dedicaría a la implementación de funciones extra, como añadir páginas de error personalizadas o un modo de juego adicional.

Semana 10: Última semana de desarrollo. Si algo quedó pendiente, evidentemente se le daría propiedad. En caso contrario, se añadiría algún sistema de logging o copias de seguridad para incrementar el número de funciones relacionadas con la administración de sistemas. También dará comienzo la redacción del documento.

Semana 11: Finalización de la redacción del documento y preparación del material para la presentación del proyecto.

Realización del proyecto

Durante los siguientes apartados se explicará en detalle cómo se ha realizado cada una de las partes que componen el proyecto, junto con las explicaciones pertinentes que justifican el uso de esas tecnologías sobre otras alternativas existentes.

Se ha dividido esta sección en los diferentes programas y herramientas utilizados y dentro de sus respectivos apartados se puede encontrar el desglose de todos los detalles concretos, de forma que sea fácil acceder a la información sobre algo en concreto sin tener que leer aquello que pueda no ser de interés.

Amazon Web Services (AWS)

Como ya se mencionó con anterioridad, todas las máquinas que se necesitan para hacer funcionar este proyecto estarán virtualizadas mediante AWS (una colección de servicios de computación en la nube pública que en conjunto forman una plataforma de computación en la nube, ofrecidas a través de Internet por Amazon).

Tras contemplar las distintas posibilidades y los límites de la capa gratuita de AWS, será necesario hacer uso de tres instancias EC2, una instancia RDS, una dirección IP elástica, un par de claves y un mínimo de dos grupos de seguridad.

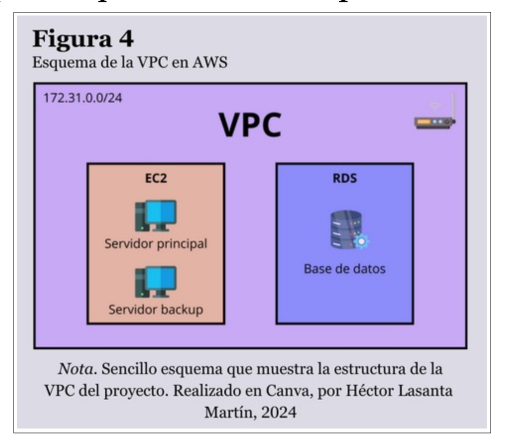


Figura 4
Esquema de la VPC en AWS

Nota. Sencillo esquema que muestra la estructura de la VPC del proyecto. Realizado en Canva, por Héctor Lasanta Martín, 2024

Las máquinas virtuales se tratarán en más detalle en los próximos subapartados, pero antes resulta importante explicar brevemente el resto de elementos.

La dirección IP elástica (asegura que la instancia tenga siempre la misma IP pública) es necesaria para que las instancias EC2 puedan utilizar el *AWS Internet Gateway* que permite la conexión entre las instancias de la VPC y el resto de internet. Esto es necesario principalmente para descargar los paquetes necesarios en ambas instancias EC2 (hay una que solo necesitará esta IP temporalmente, por eso se necesitan 2 direcciones IP elásticas) así como para permitir que los usuarios puedan conectarse al servidor principal que albergará la aplicación web. Aunque no es estrictamente necesario que la IP pública del servidor sea una IP elástica, si el objetivo es asociar dicha IP a un nombre de dominio mediante DNS, no tendría ningún sentido usar una

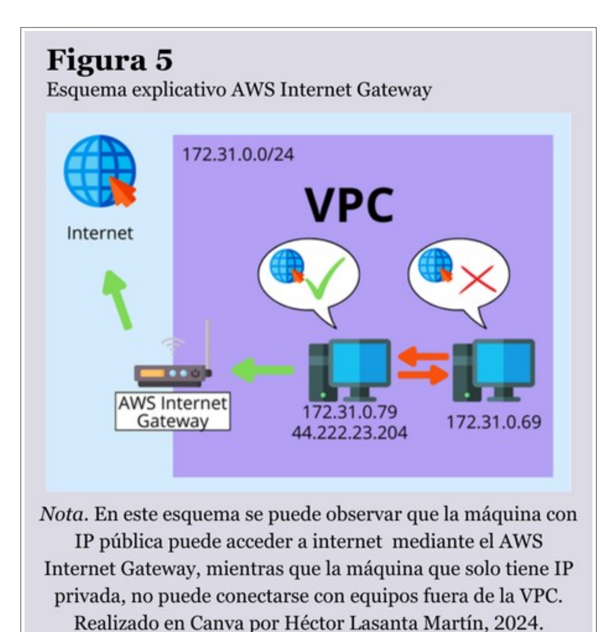


Figura 5
Esquema explicativo AWS Internet Gateway

Nota. En este esquema se puede observar que la máquina con IP pública puede acceder a internet mediante el AWS Internet Gateway, mientras que la máquina que solo tiene IP privada, no puede conectarse con equipos fuera de la VPC. Realizado en Canva por Héctor Lasanta Martín, 2024.

IP pública no elástica, ya que esta podría variar a lo largo del tiempo, causando muchos problemas.

El par de claves será necesario para conectarse remotamente a las instancias EC2, y aunque lo correcto sería generar un par de claves nuevo, se utilizará el par de claves *vockey* por defecto ya que funciona igual que cualquier otro.

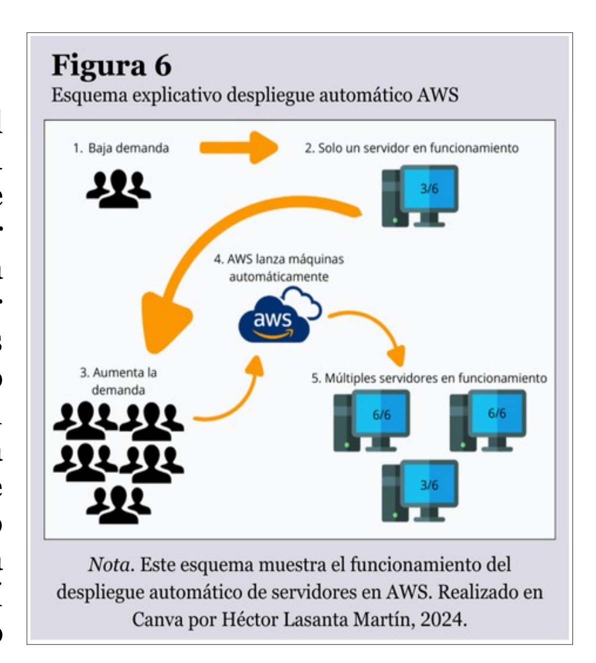
Los dos grupos de seguridad que se tendrán que configurar son el grupo para la instancia RDS y el grupo para las instancias EC2.

Instancias Elastic Cloud Computing (EC2)

Una instancia EC2 de AWS es una máquina virtual en la nube que puede configurarse de acuerdo a las necesidades existentes y a la que se puede acceder remotamente mediante SSH para trabajar en ella. Generalmente, el coste que supone una instancia EC2 es menor a lo que costaría comprar y mantener un equipo real de características similares, especialmente porque solo es necesario pagar por el tiempo que los servidores estén activos. Esto las convierte en una opción perfecta para un proyecto como este si fuera a implantarse realmente, ya que permitirían ajustar el número de servidores disponibles en cada momento para satisfacer la demanda esperada, evitando así desperdiciar recursos en mantener funcionando máquinas innecesarias.

Figura 6

Esquema explicativo despliegue automático AWS



Nota. Este esquema muestra el funcionamiento del despliegue automático de servidores en AWS. Realizado en Canva por Héctor Lasanta Martín, 2024.

Dicho esto, solamente se crearán dos máquinas virtuales, ya que crear un sistema de servidores que se ajusta a la demanda automáticamente podría ser un proyecto en sí mismo y por mucho que el autor quiera añadir un número ilimitado de funciones al proyecto, a veces hay que saber cuando es necesario parar y centrarse en mejorar lo que ya se tiene.

Ambas instancias EC2 son exactamente iguales; tienen los mismos componentes físicos, comparten el mismo grupo de seguridad y están en la misma subred. Como se mencionó con anterioridad, la única diferencia entre ambas es la existencia de una IP pública elástica asociada permanentemente a una de ellas (El servidor principal).

La configuración seleccionada a la hora de lanzar estas instancias EC2 fue la siguiente:

- **Sistema operativo:** Debian 12 con arquitectura de 64 bits (x86).
- **Tipo de instancia:** t2.micro. Es el tipo de instancia por defecto. Cuenta con un procesador poco potente y 1GiB de RAM. Si el servidor fuera a utilizarse para una aplicación web real, sería necesaria una máquina más potente, pero para el

desarrollo de este proyecto, la t2.micro es ideal ya que permite trabajar sin complicaciones por un coste de 0,0116$ / hora.

- **Par de claves:** Se podría crear uno nuevo, pero no es necesario, así que se usará el par de claves predeterminado (*vockey*) como se dijo antes.

- **Configuraciones de red:** Como no se había creado el grupo de seguridad con anterioridad, desde aquí habrá que crear uno personalizado que permita el tráfico HTTP, HTTPS y TCP al puerto 8000 (la aplicación de Django) desde cualquier dirección IP para que los usuarios puedan acceder a la página desde sus dispositivos. También se configuraron reglas de entrada para permitir el SSH y todo el protocolo ICMPv4 solamente si su origen era la dirección IP pública del autor para que pudiera conectarse remotamente a las instancias EC2 y poder hacer pruebas de conectividad de red.

- **Configurar almacenamiento:** Con la configuración predeterminada (8GiB) hay almacenamiento más que sobrado para lo poco que será necesario.

- **Detalles avanzados:** No es necesario, pero sí preferible, añadir dentro de "Datos de usuario" el comando alias ls="ls –color" para que se muestren los colores de la consola cuando se trabaja como el usuario *root* sin tener que especificarlo manualmente cada vez que se inicia la máquina.

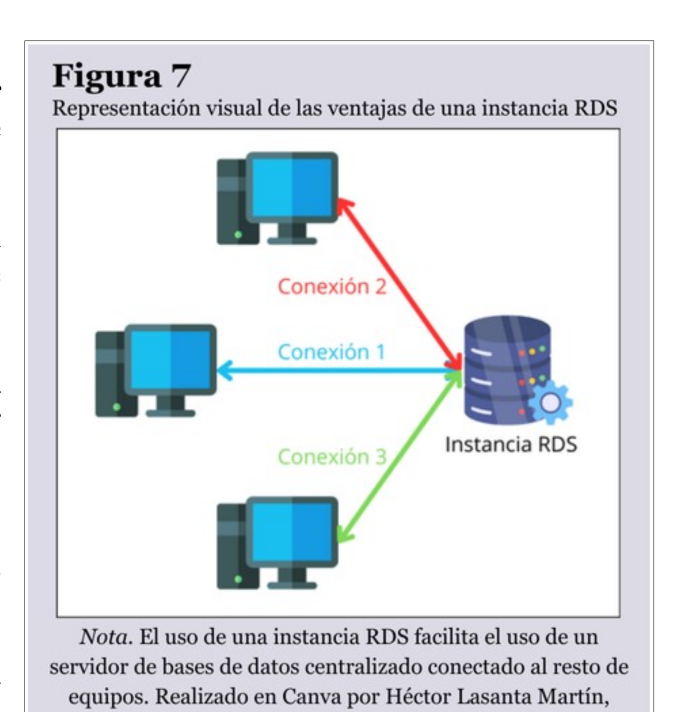
Instancia Relational Database Service (RDS)

Originalmente, la base de datos del proyecto se encontraba en el propio servidor principal. Para una aplicación pequeña que no reciba una carga de trabajo elevada esa puede ser una alternativa viable, pero ya que se está trabajando en AWS sería buena idea crear una instancia separada exclusivamente para albergar la base de datos.

Una instancia RDS permite desplegar una base de datos independiente del servidor pero dentro de su misma subred, de forma que se podrían configurar varios servidores para que todos usaran esta instancia como base de datos. Esto sería especialmente útil en casos como los que se propusieron antes: sistemas automatizados que activan o desactivan servidores dependiendo de la demanda existente en una hora y día concretos.

Figura 7

Representación visual de las ventajas de una instancia RDS



Nota. El uso de una instancia RDS facilita el uso de un servidor de bases de datos centralizado conectado al resto de equipos. Realizado en Canva por Héctor Lasanta Martín, 2024.

Aunque el proyecto no necesite esto, resulta una forma perfecta de añadirle cierta complejidad a la vez que se adquieren nuevos conocimientos y se ponen en práctica.

El proceso de creación empleado será el estándar, configurando la instancia de la siguiente manera:

- **Tipo de motor:** MariaDB versión 10.11.6. En un principio se contempló la posibilidad de utilizar PostgreSQL, pero finalmente se decidió que era preferible trabajar con MariaDB ya que el grado de familiarización del autor con este era mayor.

- **Plantilla:** Capa gratuita. Sería preferible no usar esta plantilla para personalizar un poco más la instancia RDS, pero no fue posible porque la cuenta de AWS ofrecida por el instituto carece de los permisos necesarios para crear una instancia siguiendo cualquier plantilla que no sea la de la capa gratuita.

- **Nombre y contraseña de usuario maestro:** Lo correcto sería utilizar unas credenciales de administrador seguras, pero por comodidad y dado el contexto del proyecto, no debería ser problemático utilizar el usuario *admin* con contraseña *123456Aa*.

- **Configuración de la instancia:** Con la configuración predeterminada es suficiente. Se trata de una instancia db.t3.micro con 2 CPUs y 1GiB de RAM. Al tener solo un servidor que podrá acceder a esta máquina, con esto se cubren las necesidades del proyecto.

- **Almacenamiento:** Habría sido ideal poder reducir el almacenamiento a 1GiB ya que no se va a utilizar prácticamente nada y podría haber ayudado a reducir los costes de funcionamiento de la instancia, pero desgraciadamente el valor mínimo es 20 GiB, así que se utilizó dicho valor.

- **Conectividad:** Una vez finalizada la creación de la instancia RDS se ofrece la opción de configurar una conexión entre esta y una instancia EC2, pero como ya han sido creadas las instancias EC2, aparecerá la opción “Conectarse a un recurso informático de EC2”, lo que conectará la instancia RDS automáticamente a las máquinas virtuales ya existentes.

  El resto de opciones mantendrán sus valores predeterminados, excepto la opción de grupo de seguridad, dónde hay que crear uno nuevo exclusivamente para la instancia RDS que permita todo el tráfico TCP dirigido al puerto 3306 (MYSQL/Aurora) desde cualquier dirección IP perteneciente a la VPC para que las máquinas EC2 puedan acceder a la base de datos.

Nginx

Nginx es un servidor web/*proxy* inverso ligero de alto rendimiento que por suerte para el plan presupuestario del autor es software libre licenciado bajo la Licencia BSD simplificada y lo que es más importante, gratuito.

De los dos servidores con los que se ha trabajado durante el curso, Nginx es probablemente el más apropiado para este proyecto, principalmente porque el autor lo controla mejor que Apache, a pesar de que este último es más fácil de configurar y por lo general da menos problemas.

A esto se suma el hecho de que durante el desarrollo de la aplicación surgió la necesidad de redirigir el tráfico del puerto 8000 al 80 para poder mostrar la aplicación en el navegador. Para conseguir esto fue necesario aplicar una configuración muy similar a la utilizada cuando durante el curso ha sido necesario hacer funcionar Nginx como proxy inverso para Apache, solo que como se verá en uno de los siguientes apartados, se trata de una configuración bastante más sencilla. Esto no habría sido posible con los conocimientos actuales de Apache del autor, así que parece que se tomó la decisión correcta.

Figura 8

Esquema de funcionamiento del proxy

Nota. El cliente se conecta por HTTP o HTTPS al servidor Nginx, que reenvía el tráfico al puerto 8000, donde se encuentra el servidor de Django que contiene la aplicación web. Realizado en Canva por Héctor Lasanta Martín, 2024.

Configuración básica

Dentro del archivo nginx.conf o preferiblemente en la carpeta sites enabled, se deberá incluir dentro de un bloque server la configuración del servidor que escuchará en el puerto 80 (HTTP) y posteriormente deberá ser retocada para acomodar el protocolo HTTPS en el puerto 443.

Dicha configuración incluye las siguientes directivas:

- **server_name:** Aquí se especifica el nombre de dominio al que debe responder el servidor. Se pueden configurar múltiples nombres de dominio o incluso direcciones IP, pero en este caso, no será necesario.

  El nombre de dominio utilizado es aiquimia.mooo.com, que como se puede observar es un subdominio de mooo.com. Este se obtuvo a través de la página web de FreeDNS ya que es un recurso gratuito con se ha trabajado durante el curso y funciona sin ningún problema.

- **location /:** Esta directiva se encarga del manejo de las solicitudes que llegan a la raíz del dominio. Dentro será necesario incluir la directiva proxy_pass, que reenviará las solicitudes HTTP al puerto 8000, en el que se encontrará la aplicación web.

- **error_page:** Se utilizará esta directiva para mostrar páginas de error personalizadas. Para este proyecto es importante controlar el error 502 *Bad Gateway*, ya que en caso contrario, la página de error predeterminada de Nginx se mostrará en su lugar, dejando un resultado poco profesional. Este error aparecerá cuando se intente acceder a la aplicación web mientras esta esté caída o en mantenimiento.

La página de error personalizada será un archivo html situado en el directorio homónimo de Nginx. Las imágenes utilizadas en este archivo se encontrarán en un directorio llamado imagenes502 para el que habrá que configurar una nueva directiva location en el archivo de configuración que establezca la raíz en la carpeta html anteriormente mencionada. De no hacerse esto, Nginx intentará mostrar las imágenes siguiendo la ruta configurada en la directiva location / y por lo tanto, no cargarán correctamente.

Figura 9
Imagen de error



Nota. Imagen que se muestra en las páginas de error. Generada con Microsoft Copilot, 2024.

Otro error que es importante controlar debido a la frecuencia con la que suele aparecerle a un usuario promedio es el *Not Found* (404), pero este no será necesario configurarlo en Nginx, ya que se utilizarán las herramientas de Django para implementarlo en la aplicación.

Proxy inverso

Como ya se mencionó en el apartado anterior, el servidor Nginx estará configurado como proxy inverso para reenviar las solicitudes que reciba de los clientes al servidor en el que se encuentra la aplicación. En un entorno de producción real, debería utilizarse un servidor de aplicaciones WSGI (*Web Server Gateway Interface*), pero esto implicaría desviarse más de la cuenta de los conocimientos adquiridos durante el grado y aumentaría significativamente la complejidad del proyecto. Es por esto que se utilizará el servidor de desarrollo incorporado que ofrece Django. Este no es especialmente robusto y carece de las medidas de seguridad que puede ofrecer un servidor en condiciones, pero para el contexto de este proyecto, es más que suficiente.

La directiva location / también incluye una serie de directivas proxy_set_header que ajustan algunos de los encabezados de la solicitud. Concretamente, Nginx le envía al servidor de Django el *host* original solicitado por el cliente, la dirección IP del cliente y una lista que contiene la IP del cliente junto con cualquier otro *proxy* que haya manejado la solicitud antes de que esta llegara al servidor Nginx.

HTTPS

El autor pensó que sería buena idea realizar la configuración por pasos. Es por esto que en primer lugar se configuró solo el HTTP en vez de empezar directamente con HTTPS. A lo largo del curso, se ha aprendido como utilizar OpenSSL para crear certificados autofirmados y utilizarlos con Nginx. El principal problema de estos certificados es que no han sido emitidos por una agencia certificadora (CA) de confianza y generalmente hacen que se muestre en el navegador un aviso de página insegura al entrar.

Para que esto no ocurra, este proyecto hará uso de un certificado de Let's Encrypt, que al ser una CA fiable, no hará que el navegador entre en pánico cada vez que un usuario acceda a la aplicación web.

Este proceso demostró ser bastante sencillo como veremos a continuación, pero antes de empezar a configurar el HTTPS es necesario conocer cuáles serán los nombres de dominio para los que se va a necesitar un certificado.

Para configurar el servidor utilizando HTTP, se utilizó el subdominio aiquimia.mooo.com, derivado del dominio mooo.com y ofrecido gratuitamente por el servicio *online* gratuito FreeDNS. Para que el proyecto quedase un poco más completo y profesional, se decidió pagar por un dominio personal. Esto se hizo a través de la web Hostalia, en la que se pagó por la propiedad del dominio aiquimia.es. También fue necesario configurar los registros DNS de Hostalia para que el nombre de dominio enviara a los usuarios al servidor en AWS en lugar de a la página web ofrecida por el servicio de *hosting*.

Tras esperar unas 24 horas a que se completara el proceso de propagación de DNS para que los registros funcionaran como se esperaba, ya podía dar comienzo la migración de HTTP a HTTPS.

Como se mencionó anteriormente, el proceso es sencillo, solamente será necesario seguir las instrucciones indicadas en la página web de Let's Encrypt, proceso que nos llevará a instalar el *daemon* (un tipo especial de programa que se ejecuta en segundo plano) llamado snapd mediante el cual instalaremos la herramienta certbot, que tras ser activada en modo Nginx, revisará los ficheros de configuración del servidor, mostrando por consola un menú desde el cuál se podrá elegir uno o varios de los nombres de dominio mencionados en el fichero de configuración para generar su correspondiente certificado. Se puede elegir si se prefiere solamente crear los

Figura 10

Certificado de aiquimia.es



Nota. Se puede observar que es un certificado real de Let's Encrypt para aiquimia.es. Captura de pantalla tomada por Héctor Lasanta Martín, 2024

certificados o en su lugar dejar que certbot los cree y los configure, que es lo que se hizo durante este proyecto.

Con esos pasos realizados, y suponiendo que el grupo de seguridad de AWS esté correctamente configurado para permitir el tráfico dirigido al puerto 443, la aplicación web debería estar funcionando con HTTPS y su certificado de Let's Encrypt.

Django

Django es un *framework* de desarrollo web escrito en Python ofrecido al público bajo una licencia BSD. Su meta fundamental es facilitar la creación de sitios web complejos proporcionando un extenso catálogo de características que simplifican la implementación de muchas de las funciones que se podrían necesitar durante el desarrollo, como la gestión de usuarios, grupos y permisos, la creación y migración de bases de datos, el manejo de *cookies* y errores, la autenticación de usuarios, etc.

Precisamente por esas facilidades se va a utilizar Django para este proyecto en lugar de otros *frameworks* para Python como Flask o Tkinter. De esta forma, es más sencillo centrarse en la administración de estos recursos sin tener que desviarse más de la cuenta para programar las distintas funcionalidades que el proyecto iba a necesitar.

Incluso con todas las características que incluye Django, al autor le faltaba lo más esencial: información. Al contrario que con Flask, que era la principal alternativa considerada para el proyecto, no se había utilizado Django durante el curso. Por lo tanto, fue necesario invertir cierto tiempo en aprender a utilizar las funciones básicas del f*ramework*.

Primeros pasos: Instalación de Django y creación de la aplicación.

Se puede instalar Django en una máquina de muchas formas diferentes. Para este proyecto se tuvieron en consideración dos sistemas de gestión de paquetes: APT (*Advanced Packaging Tool*) y pip (*Pip Installs Packages*).

APT es el programa responsable de la administración de paquetes desarrollado por el proyecto Debian. Facilita la gestión de software en sistemas informáticos parecidos a Unix (como GNU/Linux) al automatizar la obtención, configuración e implementación de paquetes de software, ya sea a partir de archivos ya compilados o a través de la compilación del código original.

Pip es un sistema para la administración de paquetes que se utiliza para instalar y gestionar software escrito en Python. Una gran cantidad de estos paquetes están disponibles en el índice de paquetes de Python (*Python Package Index*, PyPI). En la página del proyecto PyPI, enlazada en el anterior paréntesis, lo describen como "un repositorio de software para el lenguaje de programación Python.".

De la forma en la que se han descrito, podría parecer que cualquiera de los dos serviría para instalar Django en la máquina virtual, y esto es correcto. Sin embargo, existe una pequeña diferencia en el ámbito de instalación utilizado por ambas gestores de paquetes.

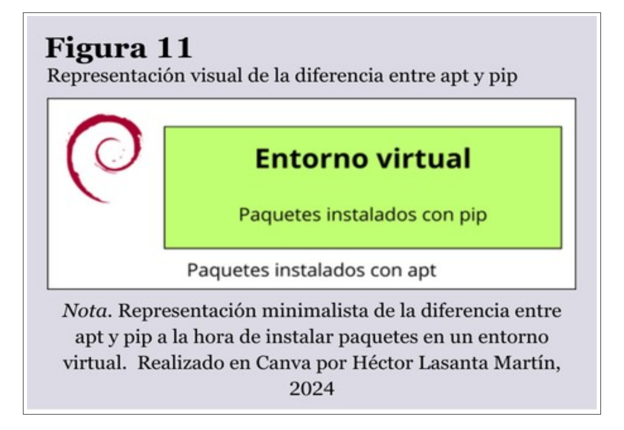
Para poder explicar esta diferencia es importante primero entender el concepto de entorno virtual. Un entorno virtual (*Virtual Environment* o VE en inglés) es es una herramienta que permite aislar las dependencias necesarias para un proyecto específico. Esto significa que cada proyecto puede tener su propio conjunto de dependencias, sin interferir con otros proyectos. El uso de un entorno virtual es considerado una buena práctica de desarrollo, principalmente para evitar conflictos entre distintos proyectos ubicados en la misma máquina.

No solo es una buena práctica, sino que algunos de los paquetes será necesario instalarlos con pip, que no permitirá hacerlo fuera de un entorno virtual. Por lo tanto, este proyecto funcionará dentro de un entorno virtual. Será necesario instalar primero venv, una herramienta de creación de entornos virtuales. Una vez creado y activado el entorno virtual mediante el uso de dos simples comandos, se podrá comenzar la instalación de Django.

Es aquí cuando resulta necesario diferenciar entre pip y apt. Al estar dentro de un entrono virtual activo, los paquetes instalados por pip solo afectarán a dicho entorno virtual, mientras que apt siempre instala los paquetes a nivel de sistema, volviendo completamente inútil el uso de un entorno virtual. Es por este motivo que a partir de ahora se utilizará pip para instalar todos los paquetes necesarios para hacer funcionar la aplicación.

Figura 11

Representación visual de la diferencia entre apt y pip



Nota. Representación minimalista de la diferencia entre apt y pip a la hora de instalar paquetes en un entorno virtual. Realizado en Canva por Héctor Lasanta Martín, 2024

Una vez instalado Django mediante pip, lo siguiente es crear la aplicación. El nombre de la aplicación creada para este proyecto es Little AIchemy (AIquimia en español), que funciona como juego de palabras combinando el nombre del juego original y las siglas IA de inteligencia artificial. En este documento no se detallarán los comandos y pasos exactos necesarios para crear y poner en marcha la aplicación. En caso de ser necesaria esta información, el autor recomienda consultar la guía de inicio ofrecida por el proyecto Django en su página web, a la que se puede acceder mediante este enlace.

Nota: La versión de Django utilizada para la realización de este proyecto es la 3.2.19.

Concluida la instalación de Django, comenzaría el desarrollo de la aplicación web. En este documento no se detallará el proceso de desarrollo desde su comienzo hasta su finalización. En su lugar, se explicará el funcionamiento de los distintos elementos que componen el proyecto ya finalizado, detallando su función y el método utilizado para su implementación. El código fuente del proyecto se encuentra en el repositorio de Github por si fuera necesario consultarlo para entender lo que se explica o por cualquier otra razón. Toda esta información se ha dividido en los apartados que pueden encontrarse justo

después de este, empezando por la configuración de Django (véase el apartado Configuración), permitiendo así acceder de manera rápida y sencilla al contenido relacionado con una característica concreta del proyecto.

Árbol de ficheros

A continuación se muestra la estructura del árbol de ficheros del proyecto, información que el autor consideró digna de mención y a la que se hará referencia a lo largo de los siguientes apartados.

Como se puede observar, todos los archivos de Python de Django del proyecto se encuentran en un directorio con el nombre de la aplicación. Además de esto, existen otros directorios que contienen los archivos estáticos. Siguiendo la estructura generalmente aceptada para la organización de los archivos, los documentos JavaScript y CSS se encuentran en el directorio *static* mientras que las páginas web en HTML o plantillas se guardan en *templates*.

Ajenos al contenido del proyecto, se encuentran los directorios extra y ssh (no presentes en la figura al tratarse de archivos no relacionados con la aplicación). Ambos se encuentran ahí por comodidad de uso durante el desarrollo y para recopilarlos en el repositorio de Github del que se hablará más adelante. Contienen archivos de configuración, *scripts* y demás utilizados por el proyecto pero independientes de la aplicación de Django.

Figura 12

Árbol de ficheros del documento

```
.
├── LittleAIlchemy
│   ├── __init__.py
│   ├── __pycache__
│   ├── admin.py
│   ├── asgi.py
│   ├── forms.py
│   ├── migrations
│   ├── models.py
│   ├── settings.py
│   ├── urls.py
│   ├── views.py
│   └── wsgi.py
├── alquimiaVenv
│   ├── bin
│   ├── lib
│   └── pyvenv.cfg
├── manage.py
├── static
│   ├── alquimia.css
│   ├── alquimia.js
│   ├── imagenes
│   └── responsive.css
└── templates
    ├── 404.html
    ├── alquimia.html
    ├── desafio.html
    ├── inicio.html
    ├── login.html
    └── registro.html
```

Nota. Resultado del comando tree -L 3 en el directorio del proyecto. Captura de pantalla realizada por Héctor Lasanta Martín, 2024

Idioma utilizado

En un principio se pensaba desarrollar la totalidad de la aplicación en español. Este plan cambió tan pronto como el autor se percató de las dificultades que el modelo de Llama2 ofrecido por Replicate y utilizado para el proyecto (más información en el apartado Vistas) presentaba a la hora de responder de la forma que se necesitaba de manera consistente. Muchas veces la IA respondía con algunas palabras en inglés, o cuando se le pedía mediante un *prompt* del sistema que respondiera con una única palabra, hacía caso omiso de la solicitud, devolviendo párrafos completos. Además, parece que escribir los *prompts* en español no solo no era de ayuda, sino que confundía más aún al modelo de lenguaje.

Teniendo eso en cuenta junto con el hecho de que el grado superior cursado por el autor es en teoría bilingüe (aunque esto se limite a dos módulos), parecía que utilizar el inglés para desarrollar el programa sería la mejor opción.

Los comentarios incluidos en el código, así como los nombres de las funciones y variables se mantuvieron en español para facilitar la comprensión del mismo en la medida de lo posible, aunque realmente habría sido mejor desarrollar el proyecto completo en inglés ya

que por convención se utiliza este idioma para todo lo relacionado con la programación y está considerado una buena práctica.

Configuración

Toda la configuración de Django se realiza en el archivo settings.py, ubicado en la carpeta de la aplicación (consultar sección árbol de ficheros del apartado anterior en caso de ser necesario).

A continuación se hablará en detalle de todas las modificaciones realizadas sobre el archivo de configuración inicial existente al crear una nueva aplicación de Django:

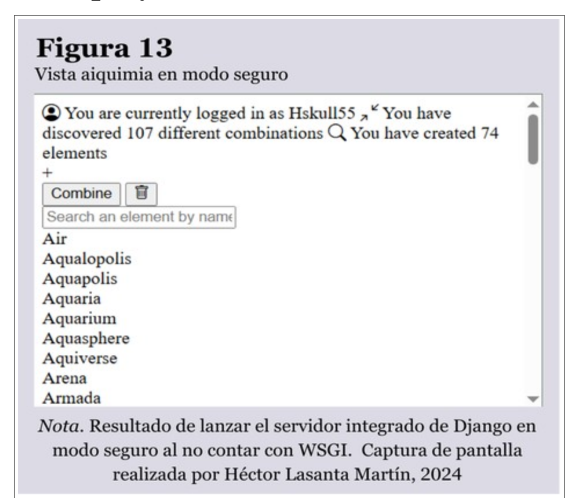
- Como se va a utilizar MariaDB para trabajar con las bases de datos del proyecto, será necesario importar el paquete pymysql haciéndolo pasar por el paquete MySQLdb, utilizado por defecto por Django.

- La clave secreta del proyecto, configurada en la variable SECRET_KEY es única para cada aplicación y se genera de forma automática durante el proceso de creación. No es necesario configurar dicha variable manualmente ya que se establece su valor de forma automática al iniciar el proyecto.

- La variable DEBUG es recomendable mantenerla en true durante el desarrollo, ya que muestra páginas de error más detalladas que ofrecen información de gran utilidad que hace bastante más fácil el proceso de búsqueda y corrección de errores.

  En teoría, el valor de esta variable deberá establecerse como false una vez finalizado el proceso de desarrollo, ya que si está activada, no será posible utilizar ciertas funciones, como el servidor WSGI o las páginas de error personalizadas configuradas desde Django. Dado que este proyecto utiliza una página de error 404 personalizada, será necesario que su valor esté en false. Esto da problemas a la hora de cargar los archivos estáticos debido a que Django espera que estos sean servidor por el WSGI, que no está presente en este proyecto como ya se ha mencionado varias veces.

Figura 13
Vista aiquimia en modo seguro



Nota. Resultado de lanzar el servidor integrado de Django en modo seguro al no contar con WSGI. Captura de pantalla realizada por Héctor Lasanta Martín, 2024

Figura 14
Vista aiquimia en modo inseguro



Nota. Resultado de lanzar el servidor integrado de Django en modo inseguro, ventana del navegador de tamaño reducido. Captura de pantalla realizada por Héctor Lasanta Martín, 2024

Para permitir que se carguen los archivos estáticos con el modo DEBUG desactivado, será necesario incluir el parámetro --insecure a la hora de activar el servidor interno de Django.

- Para poder acceder a la aplicación web, hay que configurar la lista de direcciones IP y/o nombres de dominio que la aplicación puede servir. Para ello, solamente tendremos que añadirlos a la lista de Python llamada ALLOWED_HOSTS.

- Para que funcionen con la aplicación algunas de las funciones como la consola de administración o la autenticación de los usuarios, será necesario añadir a la lista INSTALLED_APPS el nombre de la aplicación.

- El mapeo de URLs del proyecto se obtiene del fichero urls.py. Para especificar esto, tendremos que establecer el valor de la variable ROOT_URLCONF como “nombreDeLaAplicación.urls”.

- Si se contara un servidor WSGI, se debería incluir en la variable WSGI_APPLICATION el nombre de la aplicación.

- La base de datos se configura mediante un diccionario de Python en el que se debe especificar el motor de la base de datos, el nombre de la base de datos, el usuario con el que se va a acceder, la contraseña, la dirección IP (o en el caso de una instancia RDS su *endpoint*) del servidor de bases de datos y el puerto utilizado.

- Para configurar el idioma y la zona horaria deseados, será necesario añadir al archivo las variables LANGUAGE_CODE y TIME_ZONE, asignándoles los valores que corresponda. También habrá que habilitar la internacionalización, la localización y el soporte para zonas horarias.

- Se tienen que añadir también las variables STATIC_URL y STATICFILES_DIRS con los valores aplicables al proyecto en cuestión. La configuración dependerá de dónde se haya creado el directorio *static*, suponiendo que se le haya llamado así. El autor recomienda consultar la documentación oficial de Django mediante este enlace si fuera necesario.

- La página utilizada por Django para mostrar el inicio de sesión deberá configurarse mediante la variable LOGIN_URL.

- Los *logs* se han configurado mediante un diccionario de Python que contiene la siguiente información:
    - Variable que deshabilita los *logs* predeterminados de Django para utilizar exclusivamente los personalizados.
    - Subdiccionario *handlers* que envía los registros al archivo aiquimia.log.
    - Subdiccionario *root* que indica el uso del *handler* anteriormente mencionado.

- Para que Django permita el acceso a páginas mediante HTTPS es necesario indicarlo definiendo la variable SECURE_PROXY_SSL_HEADER y asignándole el valor ('HTTP_X_FORWARDED_PROTO', 'https').

Vistas

Las vistas de Django son funciones o clases de Python que Django utiliza para mostrar las páginas web de una aplicación. Generalmente, cada vista cumple una función específica y tiene su propia plantilla HTML. Nuevamente, el autor recomienda consultar la documentación oficial de Django si se precisa más información sobre las vistas, a través de este enlace.

Las vistas de una aplicación Django se configuran en el archivo views.py, ubicado dentro de la carpeta de la aplicación.

A continuación se detalla el contenido del fichero views.py de la aplicación AIquimia. Se sugiere consultar el repositorio de Github del proyecto si se desea más información al respecto de cualquiera de los elementos aquí explicados.

Importaciones

Para el correcto funcionamiento del proyecto es necesario importar varios módulos y clases, principalmente del propio Django. Entre ellos se incluyen los siguientes:

- El módulo os que permite interactuar con el sistema operativo, fundamental para el manejo de ficheros.
- El módulo datetime, necesario para registrar correctamente la hora al hacer los logs.
- El módulo random para generar números aleatorios necesarios para el modo desafío (se explicará más adelante).
- El módulo logging para poder trabajar con los logs configurados anteriormente.
- El módulo replicate para poder hacer uso de su API dentro del código.
- La clase Counter del módulo collections, utilizada para contar el número de elementos en una secuencia (en este caso se utiliza para obtener la lista de jugadores que deben mostrarse en la tabla de clasificación).
- Un gran número de módulos de Django necesarios para hacer funcionar la aplicación.
- Los modelos de la base de datos que se explicarán en un apartado posterior.
- Los formularios creados por el usuario en en archivo forms.py, si los hubiera.

También será necesario crear una variable e inicializarla con el valor del logger configurado anteriormente para poder utilizarlo en las vistas.

Ahora se explicará de forma general el contenido y funcionamiento de cada una de las vistas utilizadas por la aplicación web, pero antes, el autor considera necesario informar de la presencia de ciertos decoradores antes de cada una de las vistas.

Los decoradores son etiquetas utilizadas en Django para agregar funcionalidades específicas a las vistas. Durante el desarrollo fue necesario hacer uso de dos decoradores, @login_required, para asegurar que solo pueda accederse a esa página si el usuario ha iniciado sesión y @csrf_protect, utilizado para proteger la página contra ataques *Cross-Site Request Forgery*, un ataque mediante el cual los ciberdelincuentes se apoderan de una sesión autorizada por el usuario para realizar actos dañinos. En un principio, no se consideró necesario proteger la aplicación ningún tipo de ataque específico, pero Django obligaba a hacer uso de este decorador para permitir la correcta visualización de las páginas, así que fue obligatorio añadirlo.

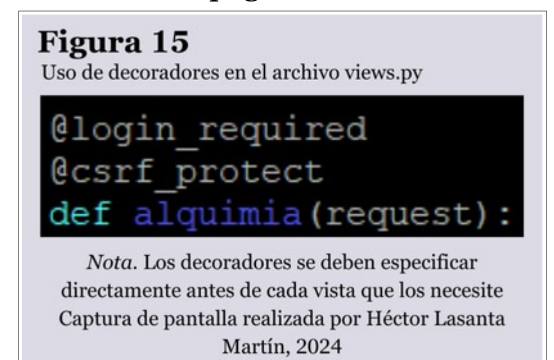


Figura 15
Uso de decoradores en el archivo views.py

Nota. Los decoradores se deben especificar directamente antes de cada vista que los necesite Captura de pantalla realizada por Héctor Lasanta Martín, 2024

Con estos conceptos explicados, es hora de analizar individualmente cada una de las vistas:

<u>Vista AIquimia (página principal)</u>

La página principal del proyecto es el lugar en el que los usuarios podrán jugar a AIquimia. Partiendo de cuatro elementos básicos (agua, tierra, fuego y aire), el jugador podrá combinarlos entre sí para generar elementos nuevos, que se añadirán a su lista de elementos en constante expansión. Desde esta página también se podrán consultar las estadísticas personales del jugador, la tabla de clasificaciones y tutorial. También se incluyen botones que redirigen al usuario a la página de inicio o al modo desafío, de cuyas respectivas vistas se hablará más adelante.



Figura 16
Página principal

Nota. Esta es la página principal de AIquimia, ventana a tamaño completo. Captura de pantalla realizada por Héctor Lasanta Martín, 2024

En primer lugar se declaran una serie de variables que se inicializan en *None* para evitar errores y se extraen todos los datos relacionados con los elementos creados por el usuario actual para posteriormente mostrarlos en el navegador.

A continuación se incluye un bloque *if* de gran tamaño que gestiona las acciones que se deben realizar en caso de que se reciba una petición de tipo POST. Esto ocurrirá cuando el usuario combine dos elementos desde la página web. Se verá la lógica detrás de ese proceso en el apartado Archivos estáticos, pero ahora se explicará lo que ocurre dentro de la vista al darse el caso.

Primero se guarda el valor de los elementos de la petición en una variable y se comprueba que se hayan enviado dos elementos. Además, se registra en el *log* la combinación realizada.

Después se compara la combinación introducida por el usuario con todas las combinaciones registradas en la base de datos. Si se detecta que dicha combinación ya existe, se obtienen sus datos y se guardan en una variable para mostrarlos al recargar la página. En caso de ser una combinación no registrada en la base de datos, se le solicita a Llama2 mediante la API de Replicate mencionada durante el apartado Importaciones que genere el nombre del elemento resultante de combinar los dos proporcionados por el usuario, así como una descripción del mismo y la imagen que se asociará al resultado. Tanto el nombre del nuevo elemento como la imagen sugerida se obtienen mediante el modelo 70b de Llama2, que es bastante preciso y avanzado. Por otro lado, la descripción se genera mediante un modelo 7b, que genera resultados de peor calidad a una velocidad considerablemente superior. Esto no es un problema porque nadie lee lo que dice la descripción de todas maneras, así que el autor consideró una desperdicio de recursos utilizar un modelo más caro y lento para generar un texto cuya presencia va a ser prácticamente imperceptible para la mayoría de usuarios.

Cuando se le solicita a la IA que sugiera una imagen, el resultado esperado es el nombre de un fichero seleccionado por el LLM de una lista que se le proporciona mediante el *prompt*. Es bastante común que la IA devuelva un párrafo completo en lugar de solamente el nombre del archivo o que lo devuelva entrecomillado / con un punto al final. Para controlar estos casos que darían lugar a error, se divide el *output* de la IA en palabras y se busca mediante patrones la que contenga ".png". Posteriormente, se eliminan las comillas y los puntos para que quede solo el nombre de la imagen.

De esta forma, se controlan la mayoría de los casos, pero a veces podría seguir dando errores. Es por esto que si tras realizar las comprobaciones anteriormente mencionadas aún no se tiene el nombre de un archivo que coincida con uno de los almacenados en el directorio imagenes, se asigna al elemento nuevo la imagen de uno de los elementos utilizados para crearlo (y se registra el suceso en el *log*), consiguiendo así que en el 100% de los casos se le asigne una imagen a los elementos que se van creando.

Cuando ya se tiene el elemento generado, se comprueba si existe otro elemento con el mismo nombre en la base de datos. En ese caso, se obtienen todos sus datos para mostrarlos al recargar. En caso contrario, se guarda el nuevo elemento en la base de datos y se establece como descubridor del mismo al usuario que lo generó, añadiendo al *log* un mensaje indicando qué usuario ha descubierto qué elemento.

Independientemente de si el elemento ya estaba registrado o no, se añade al usuario como dueño de dicho elemento y de la combinación utilizada para generarlo. Esta información es utilizada para mostrar las estadísticas de juego.

Si en algún momento se produce un error no controlado, se mostrará un mensaje de error y se registrará en el *log*.

Con esto termina el código encargado de manejar las solicitudes POST. El código restante debe ejecutarse siempre que se carga la página para poder mostrar por pantalla cierta información.

En primer lugar, se calcula el valor de ciertas estadísticas y se asigna su valor a una variable. Nuevamente, estos datos se utilizarán para mostrar las estadísticas y el marcador.

Figura 17
Thalasor



Nota. Thalasor, personaje que guía al usuario a través del tutorial. Imagen generada con Microsoft Copilot, 2024

Después se comprueba si el usuario ha completado ya el tutorial. Esto se hace mediante *cookies* que se mantienen entre sesiones. Si el tutorial no se ha mostrado previamente, se asigna a la variable tutorial el valor *true*, lo que hará que se muestre mediante JavaScript al cargar la página (Para más información consultar el apartado Archivos estáticos).

Por último, se renderiza la página con los valores de las variables que correspondan dependiendo de si se había hecho o no el tutorial.

Vista desafio (Modo desafío)

El modo desafío es un modo secundario añadido para ofrecer una modalidad de juego alejada del *sandbox* que ofrece el modo principal, pero manteniendo la esencia y las mecánicas de juego del modo principal.

En el modo desafío, el programa elegirá aleatoriamente un elemento de la lista completa de elementos descubiertos por el conjunto de los usuarios, excluyendo los cuatro elementos básicos. Después, el usuario deberá crear el elemento seleccionado en el menor número de intentos posible, teniendo a su disposición todos los elementos registrados en la base de datos.

Para implementar este modo, se reutilizó gran parte del código de la vista AIquimia. A continuación se detallan los cambios que diferencian al modo desafío del principal:

Para empezar, es necesario extraer la lista completa de elementos de la base de datos, lo que es bastante sencillo. Surge una pequeña dificultad a la hora de excluir los elementos básicos, pero mediante "objetos Q" de Django se puede hacer. Los "objetos Q" se pueden utilizar junto con los operadores & (AND) y | (OR), permitiendo así incluir el nombre de los cuatro elementos básicos en una sola consulta. El autor de este proyecto tiene un

conocimiento superficial de estos "objetos Q", solamente suficiente como para poder hacer esta consulta. Si se busca comprender en profundidad su funcionamiento, sería muy recomendable consultar el apartado "*Complex lookups with Q objects*" en la página "*Making queries*" de la documentación oficial de Django, a la que se puede acceder mediante el siguiente enlace.

Otra de las variables iniciales que actuará de forma diferente es la que guarda el valor del usuario. Para evitar que un jugador que apenas ha jugado al modo principal entre en el modo desafío y empiece a experimentar con todos los elementos ya desbloqueados, llevándose el crédito por descubrimientos no merecidos, cualquier acción que tenga lugar en la página del modo desafío le será atribuida al usuario *challenger*. Es por esto que el valor de la variable usuario se obtendrá buscando en la base de datos de usuarios de Django los valores correspondientes al usuario cuyo nombre es *challenger*.

Después aparece de nuevo el condicional que controla las peticiones tipo POST. Esta parte del código se mantiene prácticamente igual, ya que es la encargada de combinar los elementos y el modo desafío necesita mantener esa funcionalidad.

Las principales diferencias entre la vista anterior y esta se pueden encontrar tras este condicional, y son las siguientes:

En primer lugar, la forma en la que se obtienen los nombres de los jugadores que deben de aparecer en el marcador es ligeramente distinta, ya que al contar con un campo de la base de datos que contiene el número de victorias de cada usuario como valor numérico (en el modo principal se usaba un valor *many to many* no numérico), solamente hay que utilizar un ORDER BY y un LIMIT 10 para conseguir los nombres de los 10 mejores jugadores.

A continuación, se crea una *cookie* llamada contador si no existía y se inicializa su valor en 0. Si la *cookie* ya estaba presente, se incrementa en 1 su valor.

Otra *cookie* llamada elementoAleatorio contiene el nombre del elemento elegido aleatoriamente por la aplicación y que el usuario debe conseguir crear. Si no se encuentra dicha *cookie*, se elige un nuevo elemento aleatoriamente y se extraen de la base de datos los valores correspondientes a su nombre e imagen. Por el contrario, si se recibió la *cookie*, simplemente se obtiene su valor para recuperar su imagen desde la base de datos.

Por último, se compara el nombre del elemento aleatorio obtenido de la *cookie* con el nombre del elemento generado por el usuario. Si ambos coinciden, el usuario ha ganado, y la aplicación hará lo siguiente:

Lo primero que se intenta es recuperar el valor de la columna victorias desde la base de datos de victorias. Si no se encuentra ninguna valor de victorias asignado al usuario actual, se establece el número de victorias en 0.

A continuación, se incrementa en 1 el número de victorias del usuario y se guarda el cambio.

Lo siguiente que hace el programa es borrar las cookies contador y elementoAleatorio para empezar una nueva partida con un elemento aleatorio diferente y que el contador se vuelva a poner a 0.

Finalmente, se crean las cookies nuevas y se renderiza la vista.

Vista inicio

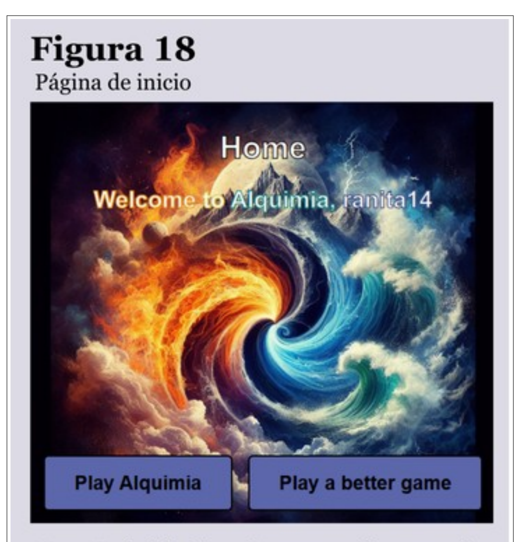
Esta vista se encarga de renderizar la página de inicio. Esta es la página que se muestra tras un inicio de sesión exitoso o al entrar en la raíz del sitio web.

Solo se realizan dos comprobaciones en esta vista.

Por un lado, se realiza una consulta a la tabla Users de Django que contiene toda la información de los usuarios registrados para comprobar si el usuario actual pertenece al grupo administradores.

Por otro lado, se comprueba el valor de la *cookie* clienteHabitual, que debería ser true si el usuario ya ha accedido previamente a la página y false si es su primera visita. En el primer caso, simplemente se renderiza la página, y en el segundo, primero se crea dicha *cookie* y luego tiene lugar el renderizado.

Figura 18
Página de inicio



Nota. Raíz del sitio web, ventana de navegador reducida. Captura de pantalla realizada por Héctor Lasanta Martín, 2024

Clase MiLoginView

LoginView es el formulario predeterminado de Django para el inicio de sesión de usuarios. En esta vista solamente se especifica la plantilla html a utilizar y se especifica que tras un inicio de sesión exitoso el usuario sea enviado a la raíz del sitio web.

Vista registro

Esta vista es la encargada de mostrar y procesar el formulario de registro. Si el método de la petición no es POST, se mostrará el formulario de registro creado en el archivo de formularios forms.py, en el que se pueden modificar los campos solicitados según sea necesario.

Por el contrario, si el método de la petición es POST, la vista esperará recibir la información del formulario de registro para comprobar su validez. Tras validar el formulario, redirigirá al usuario de vuelta a la página de inicio de sesión. Si el formulario no es válido, se volverá a renderizar la página de registro y se informará al usuario del problema durante la validación.

Vista logout

Esta vista utiliza la función logout de Django para cerrar la sesión del usuario actual. Una vez hecho esto, se le redirige a la página de inicio de sesión.

Vista custom404

Mediante esta vista se indica a Django que plantilla debe renderizar cuando se produzca un error 404: Not Found.

Modelos

Los modelos de Django, configurados en el archivo llamado models.py, son en esencia tablas de la base de datos del proyecto que se detallan en dicho fichero para que Django pueda trabajar con ellas, de forma que sean accesibles para cualquier parte del proyecto que las necesite.

Definir la estructura de las tablas de esta manera hace que sea muy sencillo trabajar con las bases de datos y sobre todo modificarlas, ya que no es necesario hacer uso de ningún sistema gestor de bases de datos. Simplemente con reflejar los cambios en el fichero y hacer las migraciones necesarias, ya deberían reflejarse los cambios en la base de datos que se haya especificado en el archivo settings.py.

Durante el desarrollo de este proyecto se han utilizado tres modelos de Django personalizados que se explicarán en detalle a continuación, así como el modelo que contiene los datos de los usuarios, que ya ofrece el propio Django y que solamente será necesario importar.

Los modelos configurados para la base de datos de AIquimia son los siguientes:

- **Elementos:** Este modelo contiene toda la información sobre cada uno de los elementos que los jugadores van descubriendo a medida que hacen las combinaciones. Guarda los siguientes datos:
    - nombre: Una cadena con el nombre del elemento.
    - descripción: Un campo de texto que contiene la descripción del elemento generada por la IA.
    - imagen: Una cadena que contiene la ruta a una imagen.
    - upload: Este es un campo de Django que permite subir archivos desde la interfaz web de administración. Originalmente se iba a utilizar para subir las imágenes, pero al final se descartó la idea porque surgieron muchos errores, lo que supuso el reemplazo de este campo por el anterior. Sin embargo, tener un botón que permitía subir imágenes desde el navegador directamente hasta el directorio de

imágenes del proyecto resultó bastante útil, así que este campo se mantuvo durante el proceso de desarrollo no para guardar datos, sino para facilitar el traslado de archivos.

- descubiertoPor: Una cadena que guarda el nombre del usuario que descubrió el elemento. *Nota:* Debió haber sido una clave foránea que hiciera referencia al modelo Users de Django, pero el autor no se percató de ese detalle hasta la redacción de este apartado, ya finalizado el desarrollo.

- creadores: Este es un campo "many to many" de Django, lo que se traduciría a una relación "muchos a muchos" en MariaDB. A pesar de que en el modelo aparece como un dato más, en realidad esta relación se guarda como una tabla intermedia entre esta tabla y la tabla Users que representa la relación.

  El campo creadores, presente en este modelo y el siguiente, sirve para registrar que elementos o combinaciones han sido creados por cada usuario y poder hacer las consultas necesarias a la hora de mostrar el "progreso" del jugador.

- **Combinaciones:** En este modelo se almacenan las combinaciones creadas por los usuarios, es decir, los dos elementos combinados y el elemento resultante. Todos los datos guardados en este modelo son claves foráneas que hacen referencia al modelo anterior, de forma que cualquier cambio sobre un elemento, así como su eliminación provocaría un efecto cascada que reflejaría dicho cambio en los datos de esta tabla.

  - elemento1: Una cadena con el nombre del primero de los elementos combinados por el usuario.

  - elemento2: Una cadena con el nombre del segundo de los elementos combinados por el usuario.

  - resultado: Una cadena que guarda el nombre del elemento resultante.

  - creadores: Su función es la misma que en el modelo anterior pero para las combinaciones.

- **Victorias:** En este modelo solo se guardan dos datos utilizados para llevar el recuento de las victorias totales de los usuarios en el modo desafío:

  - nombre: Cadena que guarda el nombre del usuario.

  - victorias: Número entero sin signo que se incrementa tras cada victoria.

Nota: Aunque aquí no se especifique, en todos los modelos existe un id autoincremental que actúa como clave primaria. Además, se especifica que los campos que no deben dejarse en blanco no permitan valores nulos. Esto puede apreciarse en el diagrama siguiente:

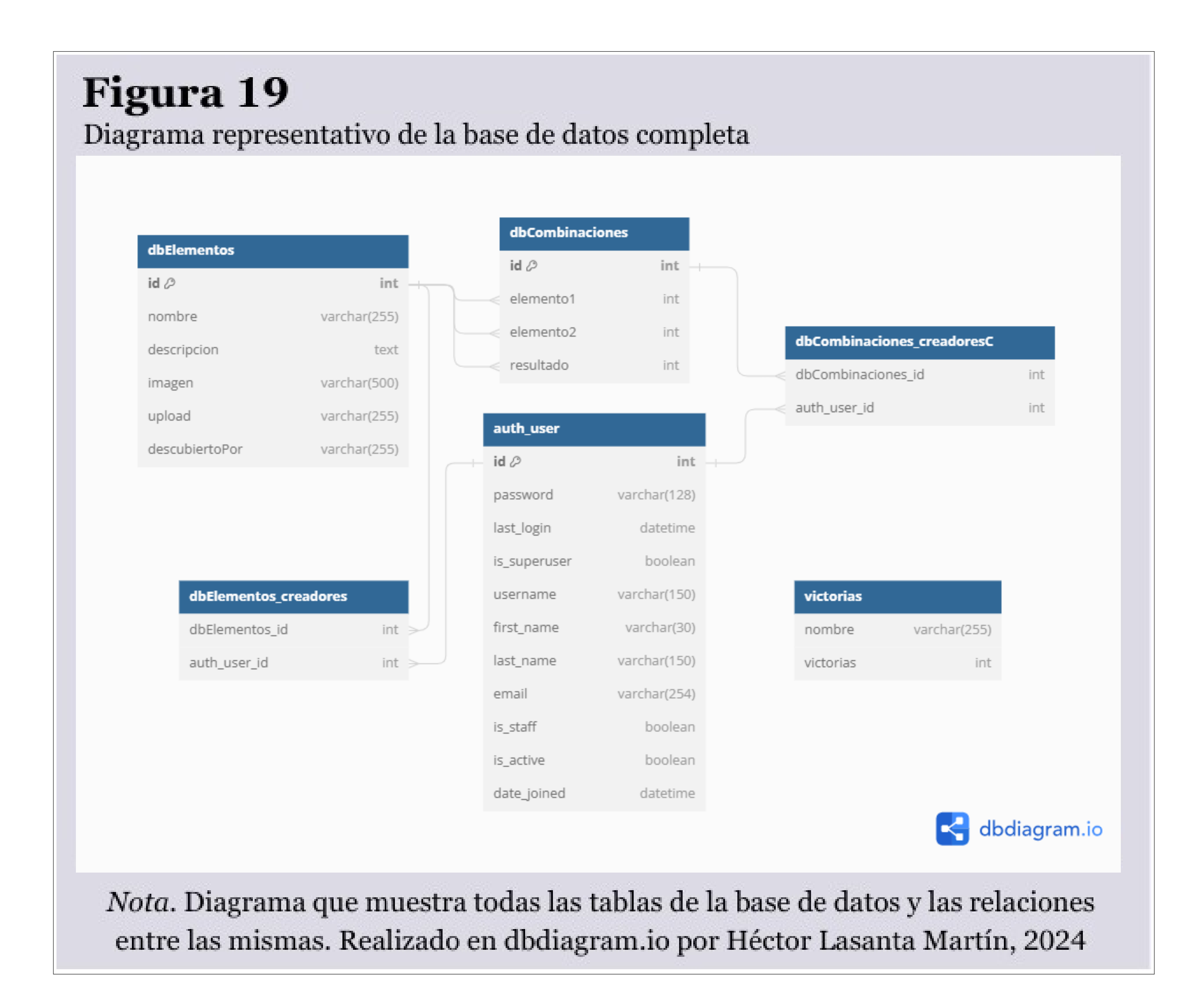


Figura 19

Diagrama representativo de la base de datos completa

Nota. Diagrama que muestra todas las tablas de la base de datos y las relaciones entre las mismas. Realizado en dbdiagram.io por Héctor Lasanta Martín, 2024

Archivos estáticos

En el ámbito de este proyecto, se entiende por archivos estáticos a las plantillas HTML y los ficheros CSS y JavaScript que hacen funcionar la aplicación web de Django.
En los siguientes subapartados se detalla el contenido de estos archivos:

HTML

Las plantillas HTML contienen el esqueleto de las páginas web del proyecto de Django que se renderizan a través de las vistas. En ellas se puede acceder a las variables enviadas desde el código Python durante la fase de renderizado. Cada vista cuenta con su correspondiente plantilla HTML, incluyendo las páginas de error personalizadas.

A continuación se detalla el contenido de cada una de las plantillas:

<u>aiquimia.html / desafio.html</u>

Ambas plantillas son prácticamente iguales, por lo que se detallará su contenido a la vez.

Dentro del encabezado es necesario que estén presentes los siguientes elementos:

- La etiqueta {% load static %} es necesaria para que Django cargue los archivos estáticos CSS y JavaScript así como las imágenes (el contenido de la carpeta static).

- Las etiquetas *script* enlazan el archivo creado por el autor y el código necesario para mostrar *pop-ups* personalizados gracias a SweetAlert.

- La ruta al icono que se mostrará en la pestaña y en la barra de marcadores (generalmente conocido como favicon)

Figura 20

Uso de la etiqueta {% load static %}

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport">
    <title>AIquimia</title>
    {% load static %}
</head>
```

Nota. Carga de archivos estáticos en plantilla HTML, encabezado simplificado. Captura de pantalla realizada por Héctor Lasanta Martín, 2024

- La ruta a la biblioteca de iconos de Bootstrap, utilizados para que todos los iconos del sitio web tuvieran un estilo consistente y homogéneo.

- Las rutas a los archivos CSS creados por el autor.

Nota: El encabezado de las siguientes plantillas incluye prácticamente el mismo contenido, generalmente menos, por lo tanto, no se volverá a mencionar.

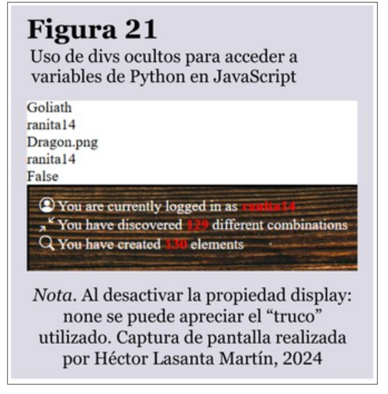
Dentro del cuerpo encontramos la siguiente estructura:

- Un gran número de divs ocultos que contendrán el valor de las variables de Python. Esto ha sido un truco empleado por el autor para poder acceder a los valores de estas variables desde el código JavaScript de manera sencilla usando la propiedad innerText.

- Todo el contenido real de la página se encuentra dentro de dos contenedores, uno para el juego y otro para la barra de navegación. El primero está dividido en dos mitades, izquierda y derecha, siendo la primera la zona donde se encuentran las estadísticas, los recuadros donde se introducen los elementos a combinar y el formulario para combinarlos (cuyos campos están ocultos ya que sus valores los obtiene el JavaScript). La parte derecha es otro contenedor en el que se muestran todos los elementos disponibles mediante un bucle que además cuenta con una barra de búsqueda que permite al usuario encontrar rápidamente el elemento que necesita cuando se vuelve complicado ubicarlo en la lista completa.

Figura 21

Uso de divs ocultos para acceder a variables de Python en JavaScript



Nota. Al desactivar la propiedad display: none se puede apreciar el "truco" utilizado. Captura de pantalla realizada por Héctor Lasanta Martín, 2024

La barra de navegación permite volver al menú principal, comprobar las tablas de clasificación, mostrar el tutorial, cambiar entre el modo principal y el modo desafío (se activa y desactiva el interruptor) y cerrar sesión.

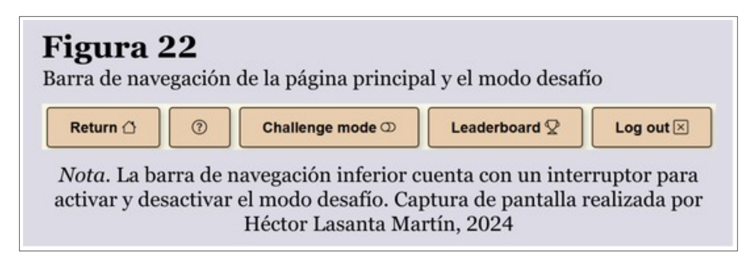


Figura 22
Barra de navegación de la página principal y el modo desafío

Nota. La barra de navegación inferior cuenta con un interruptor para activar y desactivar el modo desafío. Captura de pantalla realizada por Héctor Lasanta Martín, 2024

inicio.html

Esta plantilla contiene un mensaje de bienvenida que saluda al usuario que ha iniciado sesión y dos botones: uno para entrar al modo principal y otro que redirige al usuario a la página de Little Alchemy, el juego original en el que se basa este proyecto.

Si el usuario pertenece al grupo "Administradores", se mostrará un tercer botón oculto para los usuarios normales que permite el acceso a la interfaz de administración web de Django.

Por último, hay un pequeño fragmento de código JavaScript incluido que muestra un *pop-up* la primera vez que se entra a la página.

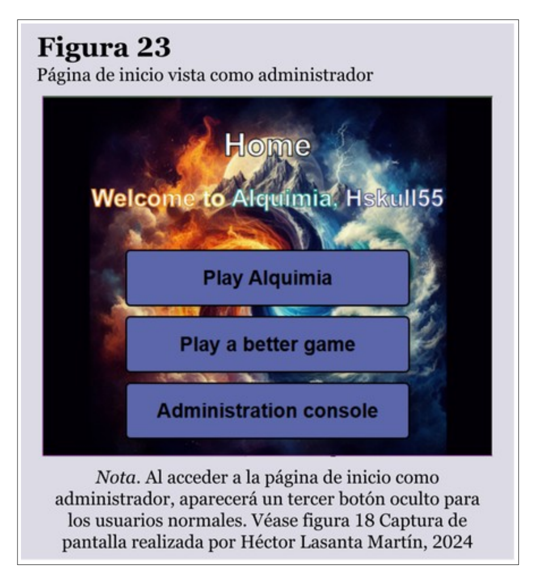


Figura 23
Página de inicio vista como administrador

Nota. Al acceder a la página de inicio como administrador, aparecerá un tercer botón oculto para los usuarios normales. Véase figura 18 Captura de pantalla realizada por Héctor Lasanta Martín, 2024

login.html

Contiene el formulario de inicio de sesión y un enlace a la página de registro por si el usuario no tiene una cuenta.

registro.html

Contiene el formulario de registro.

404.html

Contiene los estilos y el mensaje que se muestran cuando tiene lugar un error 404: Not Found.

CSS

Los estilos del proyecto se dividen principalmente en dos ficheros diferentes, uno que contiene los estilos de todas las páginas y otro que maneja la responsividad de la página al reducirse el tamaño de la ventana del navegador. A continuación se resumen ambos:

aiquimia.css

Es la hoja de estilos principal del proyecto. No se explicarán los estilos concretos aplicados, así que si fuera preciso se pueden consultar en el repositorio de Github, accesible mediante este enlace.

Para organizar los contenedores de las plantillas y su contenido se utilizó la funcionalidad de CSS conocida como flexbox. También se contempló la posibilidad de utilizar las clases proporcionadas por Bootstrap, un framework que contiene multitud de plantillas predeterminadas para elementos HTML y estilos CSS, pero el sistema de flexbox resultó ser más compatible con los métodos utilizados por el autor para diseñar los estilos, así que Bootstrap se acabó descartando.

Además de estilos básicos, también se encuentran en este archivo estilos que se aplican al situar el cursor encima de ciertos elementos (:hover), al recibir el foco de la aplicación (:focus) y animaciones que muestran un parpadeo multicolor alrededor de los *pop-ups* de SweetAlert.

Figura 24

Ejemplos de estilo usando :hover



Nota. La pseudoclase :hover permite añadir estilos que se aplican al poner el cursor sobre un elemento. Capturas de pantalla realizadas por Héctor Lasanta Martín, 2024

responsive.css

Este archivo se encarga de ocultar elementos de la interfaz de usuario no esenciales para garantizar que la página se muestra correctamente al reducir el tamaño de la ventana del navegador. Si esta se hace excesivamente pequeña, se ocultarán todos los elementos para evitar la superposición de los mismos y se mostrará un fondo con estrellas.

JavaScript

Como se ha mencionado en el apartado anterior, los *pop-ups* personalizados se han creado gracias a SweetAlert, una colección de herramientas de JavaScript gratuitas que ofrece una alternativa a los *alert* predeterminados de JavaScript que por lo general no quedan bien en casi ninguna página web y generan una mala impresión en el visitante.

Por otro lado, salvo por pequeños fragmentos de código presentes en las plantillas HTML, todo el código JavaScript escrito por el autor del proyecto se encuentra en el fichero aiquimia.js, cuyo contenido se detalla a continuación:

- Un bucle que busca todos los elementos HTML con clase elemento y les añade el *event listener* necesario para que se puedan arrastrar de un contenedor a otro así como la clase arrastrando, que tiene ciertos estilos aplicables solo a los elementos mientras son arrastrados. Para mostrar el elemento durante el arrastre se crea un clon del elemento original.

- Un *event listener* que controla lo que ocurre cuando se deja de arrastrar un elemento. Si el elemento clonado se arrastra sobre un contenedor vacío, se añade al mismo, pero si no es el caso, se elimina el clon.

- Un *event listener* que se añade a todos los contenedores mediante un bucle foreach para revisar que los contenedores estén vacíos al arrastrar un elemento sobre ellos. El bucle también añade a todos los contenedores un evento de click para eliminar un elemento de un contenedor al hacer click sobre el este.

- Un *event listener* que mueve los elementos a un contenedor vacío tras hacer click sobre ellos.

- La función combinar, que obtiene los ids y nombres de los elementos combinados para asignárselos a los campos ocultos del formulario descrito en la plantilla aiquimia.html. Finalmente, envía el formulario con estos datos para que sea procesado por el código Python.

- La función mostrarTutorial, que muestra al usuario el tutorial a partir de un array que guarda el texto de cada página del mismo.

- Se utiliza una variable que guarda un valor numérico asociado a cada una de las páginas del tutorial. Mediante SweetAlert se mostrará la página 0 del tutorial usando la imagen "tutorial0.png" y el texto en la posición 0 del array. El *pop-up* mostrará dos botones, uno para avanzar y otro para retroceder, esto hará que el valor de la variable que actúa como contador se incremente o disminuya, cambiando así la página del tutorial que se muestra. Retroceder en la primera página del tutorial permitirá salir del mismo.

- Una función que se ejecutará al cargarse por completo la página que mostrará el *pop-up* que corresponda dependiendo de las circunstancias (nuevo elemento creado, victoria del modo desafío o error al generar la combinación).

Figura 25

Pop-up del tutorial



Nota. Ejemplo de pop-up personalizado creado con SweetAlert y CSS. Captura de pantalla realizada por Héctor Lasanta Martín, 2024

- Una función que obtiene el contenido del marcador y lo muestra mediante un *pop-up*.

- Una función que elimina todos los elementos que se encuentren dentro de un contenedor, utilizada para el botón de eliminar con un icono de una papelera.

- Una función que se ejecuta cuando el documento HTML (sin importar el resto) se ha cargado completamente que se utiliza para hacer funcionar la barra de búsqueda que se encuentra dentro del contenedor de elementos.



Figura 26
Funcionamiento barra de búsqueda

Nota. Ejemplo de uso de la barra de búsqueda. Captura de pantalla realizada por Héctor Lasanta Martín, 2024

Esta función añade un *event listener* que se activa cada vez que se suelta una tecla, lo que permite que la barra de búsqueda muestre automáticamente los resultados coincidentes a medida que el usuario escribe, sin necesidad de hacer click en ningún botón para buscar. Los resultados no coincidentes se ocultan mediante la propiedad display con el valor none.

URLs

En el archivo urls.py de una aplicación Django se especifican las relaciones entre las vistas del proyecto y la URL en la que se muestra dicha vista. En este archivo se pueden configurar las URLs de muchas maneras y usando multitud de herramientas, pero en este proyecto solamente se usará el archivo para asociar las rutas con el nombre de las vistas. Si resultara interesante indagar sobre el resto de funciones que ofrece Django con respecto a la configuración de este fichero, no sería mala idea echarle un vistazo a la página correspondiente de su documentación oficial, a la que se puede llegar haciendo *click* aquí.

En el archivo va a ser necesario importar los módulos de Django relacionados con el inicio de sesión y la autorización de usuarios, el módulo de administración y las vistas de Django que se hayan configurado.

Dentro del bloque urlpatterns es donde se deberán configurar las rutas de todas las vistas del proyecto. Para conseguir esto, se debe especificar la ruta relativa a la raíz del proyecto, la vista que se está configurando, y el nombre que se le va a dar. Esto último es para poder hacer referencia a la vista con facilidad en otros lugares del código.

Además de esto, si se quieren usar páginas de error personalizadas, es importante especificar en este archivo las rutas a las vistas para las mismas. Para hacerlo, se debe incluir un *handler*X dónde X es el código de error que se quiere manejar. Para es caso de este proyecto, X será 404, que es la única página de error personalizada gestionada por Django en el proyecto (recuérdese que el error 502 lo controla Nginx).

El *handler* deberá contener una directiva *path* con la ruta a la vista que le corresponda, igual que cualquiera de las definidas en urlpatterns.

Administración

Una de las funciones más útiles y cómodas de las ofrecidas por Django es la consola de administración web que se puede añadir al proyecto y que supone una enorme mejora de la calidad de vida a la hora de desarrollar una aplicación web con Django.

Para poder incluirla al proyecto es necesario realizar un par de pasos previos.

En primer lugar, será necesario asegurarse de que la ruta al sitio web de administración, por ejemplo admin/, esté definida en el archivo urls.py explicado con anterioridad. La configuración por defecto de este archivo incluye la ruta, pero es importante asegurarse por si se ha modificado durante el desarrollo.

El segundo paso es configurar el archivo admin.py, en el que se deben importar los modelos de nuestro proyecto para poder registrarlos en la página de administración, lo que permitirá el manejo de las tablas de la base de datos desde la interfaz web gráfica, lo que resulta muy útil durante las fases iniciales del desarrollo cuando hay un número elevado de errores y *bugs*.

Por último, será necesario crear un usuario que tenga el estado de "personal" asociado, lo que viene a ser una cuenta de administrador. Esta cuenta puede crearse con el archivo manage.py y el comando createsuperuser.

Una vez se hayan cumplido todos los pasos, se podrá acceder a la interfaz de administración web a través de la ruta especificada en el archivo urls.py, siempre que el servidor esté activo y funcionando.

La interfaz web, cuyo idioma depende del configurado para la aplicación en el fichero settings.py, solo será accesible para los usuarios con permisos de administración, ya que al resto de usuarios les saltará un mensaje de error impidiendo el acceso.

Una vez dentro, no solo será posible gestionar la base de datos y modificar las tablas, sino que además se podrán crear usuarios y grupos a los que se les podrán asociar los permisos que resulten pertinentes mediante una lista de control de acceso (ACL). Para experimentar con esta función en este proyecto se han creado multitud de usuarios así como el grupo administradores, que cuenta con todos los permisos y al que pertenecen solo dos de los usuarios. De esta forma ha sido posible comprobar el funcionamiento de esta consola de administración, así como poner a prueba ciertas funciones ya explicadas en el apartado Archivos estáticos.



Figura 27
Consola de administración web

Nota. Página de administración de Django, ventana de navegador reducida. Captura de pantalla realizada por Héctor Lasanta Martín, 2024

Scripts de bash

Para poder automatizar los procesos de creación y almacenamiento de copias de seguridad de la base de datos de la aplicación así como de los *logs* de la misma, se han utilizado unos *scripts* de bash bastante sencillos que se ejecutan diariamente a una hora concreta mediante trabajos de cron configurados en ambas instancias EC2.

El autor tiene constancia de que se podrían haber combinado varios de estos *scripts* en uno solo, pero considera que es preferible tenerlos separados aunque cada uno contenga poco más de un comando para que sea aún más fácil realizar modificaciones concretas sobre ellos.

Todos los *scripts* usan la variable especial $? para comprobar si se ejecutaron correctamente y mostrar un mensaje en la consola indicando si fue así o no. Esto en principio no sería necesario ya que los *scripts* se ejecutan automáticamente y no manualmente por un usuario, pero se pensó que no sería mala idea añadir esta función por si alguna vez ocurriera esto último.

Los *scripts* podrían resumirse de la siguiente manera:

- **backup.sh:** Se encuentra en la instancia EC2 dedicada a almacenar la información de respaldo y los *logs*. Básicamente hace un mysqldump de la base de datos ubicada en la instancia RDS y guarda el fichero resultante en la carpeta /dbBackup incluyendo en el nombre del fichero generado la fecha y la hora exactas en las que fue creado.

- **moverLogs.sh:** Este *script* se encarga de mover los *logs* generados por Django en la carpeta donde se encuentra la aplicación web al directorio temporal (/tmp) para que el siguiente *script* pueda acceder a ellos y copiarlos.

- **copiarLogs.sh:** Este *script* está en la máquina de *backup* y su finalidad es copiar los *logs* que se encuentran en el directorio temporal a un directorio específico dentro de la máquina de respaldo. De esta manera, trabaja en conjunto con el *script* anterior; el primero mueve los archivos al /tmp del servidor y este los copia desde allí hasta el directorio donde quedarán recopilados todos los *logs* por si fuera necesario acceder a alguno de ellos en cualquier momento.

- **limpiarTmp.sh:** Borra el contenido del directorio temporal del servidor una vez al día. Esto es necesario si se piensa dejar el servidor en funcionamiento todo el día como podría esperarse de un servidor en un caso real, ya que en Debian, el directorio tmp solo pierde su contenido al iniciarse el sistema operativo. Esto implica que si no se vaciara regularmente el directorio mediante un *script* o algo similar, se empezarían a acumular todos los *logs* en el /tmp, siendo esto algo que es preferible evitar para mantener el flujo de archivos lo más limpio posible.

Repositorio de Github

Durante la fase de planificación del proyecto se llegó a la conclusión de que trabajar en este proyecto era la oportunidad perfecta para aplicar los conceptos teóricos sobre git y Github aprendidos durante el primer año del grado. No solo porque son herramientas que cualquier informático debería conocer y controlar, sino porque desarrollar una aplicación web sin contar con algún tipo de control de versiones sería una decisión demasiado arriesgada que inevitablemente terminaría resultando en una pérdida de tiempo a largo plazo.

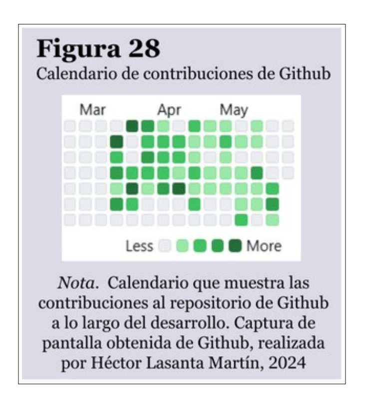
Es por esto que el autor decidió crear un repositorio privado de Github (en estado público desde que finalizó el desarrollo y accesible mediante este enlace) para clonarlo en la máquina Debian y así poder ir subiendo *commits* cada vez que se consiguiera implementar un cambio significativo, de forma que el proyecto se mantuviera estable y libre de *bugs* en la medida de lo posible.

Para poder clonar el repositorio fue necesaria la creación de un token de acceso personal con permisos de control total y que no expirara nunca.

Un día de desarrollo normal implicaba la adición de una nueva mecánica o funcionalidad al proyecto y su(s) respectivo(s) *commit*(s). En algunas ocasiones, y principalmente para aprender más sobre el uso de git, en lugar de implementar una nueva funcionalidad directamente en la rama principal, se creaba una rama separada dónde experimentar sin tener que preocuparse por la integridad del código existente. Una vez que la rama había alcanzado un estado en el que las nuevas funciones añadidas al proyecto eran lo suficientemente estables y compatibles con la rama principal, se hacía un pull request al autor del proyecto y se fusionaban ambas ramas.

Figura 28

Calendario de contribuciones de Github



Nota. Calendario que muestra las contribuciones al repositorio de Github a lo largo del desarrollo. Captura de pantalla obtenida de Github, realizada por Héctor Lasanta Martín, 2024

El repositorio de Github también contiene un archivo README escrito en markdown que se ha utilizado como "webgrafía", un directorio llamado extra que contiene todos los archivos no relacionados directamente con el código fuente de la aplicación web (ficheros de configuración, claves de SSH, plantillas html de las páginas de error, archivos de *log*, *scripts*, etc.) y otro archivo, LICENSE, que indica el tipo de licencia aplicada al proyecto.

El directorio extra solamente se ha incluido en el repositorio para que ciertos archivos importantes del proyecto sean fácilmente accesibles para cualquier persona que lo esté leyendo y / o esté interesada en él. Dejar ficheros con las claves SSH o los tokens usados por las APIs es extremadamente inseguro y bajo ningún concepto debería realizarse en un caso real. Nótese que la información visible en el repositorio de Github a la hora de hacerse público el mismo era distinta a la utilizada realmente durante el desarrollo del proyecto y solo está ahí a modo de ejemplo.

La licencia que utilizada para el proyecto es la conocida como *Unlicense* o "La no licencia", que indica que el proyecto se trata de software gratuito lanzado al dominio público, autorizando a cualquier persona que así lo desee copiarlo, modificarlo, publicarlo, usarlo, compilarlo, venderlo o distribuirlo. A su vez, la licencia indica que el *software* se proporciona sin ninguna garantía y que sus autores no serán responsables de ningún problema que pueda ocasionar.

La decisión del autor fue marcada por el nombre de la licencia, que llamó su atención al ser mínimamente gracioso al contrario que el resto. Una vez seleccionada la licencia, el autor se dio cuenta de que estaba totalmente de acuerdo con lo que en ella se decía y por eso la mantuvo como licencia definitiva.

Modificaciones sobre la propuesta de proyecto original

La propuesta de proyecto original no llegó a ser revisada por el profesorado hasta pasadas varias semanas de la fecha de finalización de la segunda evaluación. Esta situación llevó al autor a pensar, probablemente de manera acertada, que el contenido propuesto inicialmente en el anteproyecto iba a ser insuficiente y requería por su parte un esfuerzo por ampliarlo para que cumpliera con lo esperado de este módulo.

Esto le llevó a introducir un gran número de cambios y nuevas funciones al proyecto que a su parecer lo elevan al nivel de complejidad requerido por el módulo. Para que quede constancia de dichas adiciones (aunque ya se explicaron anteriormente junto con el resto de la aplicación), se han recopilado en la siguiente lista:

- **Sistema de copias de seguridad y *logging*:** Uno de los principales motivos por los que se decidió implementar estas funciones al proyecto fue para poder ampliar el número de funciones relacionadas directamente con lo aprendido durante los dos años cursados. Estos añadidos junto con la instancia RDS, que en un principio parecía que iba a ser imposible de configurar, supusieron un cambio drástico en la estructura del proyecto, que pasó de funcionar en una instancia EC2 a necesitar una VPC completa con tres máquinas interconectadas.

  Fue necesario asociar una dirección IP elástica temporalmente a la nueva instancia EC2 para poder descargar los paquetes requeridos para conectarse a la instancia RDS (mariadb-server) y configurar la automatización de los *scripts* pertinentes (cron).

  Una vez configurado, este sistema ayuda en gran medida a detectar errores y compararlos con lo sucedido en situaciones similares anteriores mediante sus *logs*; a la vez que proporciona un registro diario del estado de la base de datos para poder restaurar el estado de un día concreto en caso de que fuera necesario por corrupción de datos, migración a otra instancia RDS / base de datos, fallo de la instancia RDS, etc.

- **Modo desafío:** La necesidad de incluir un modo secundario que se alejara del concepto de "caja de arena ilimitada" para ofrecer una experiencia con objetivos más claros estuvo presente desde la concepción del proyecto como idea. Varios miembros del profesorado indicaron la quizás excesiva simpleza a nivel interactivo que una aplicación web como la que se propuso podría implicar. Sus comentarios llevaron al autor a idear una serie de mejoras que podrían añadir cierta interactividad al proyecto. De ahí surgió la idea de añadir este modo, entre otros aspectos de menor importancia de los que se hablará en apartados posteriores a este.

  Ante la preocupación generada en algunos profesores por el hecho de que este nuevo modo podría considerarse básicamente un juego nuevo y por ende una exigencia demasiado elevada dado el reducido tiempo restante antes de la fecha de

entrega del proyecto, el autor mantuvo la creencia de que un añadido de estas características era en realidad bastante sencillo de implementar.

Como ya se ha explicado en su propio apartado, el modo desafío se desarrolló trabajando sobre la base del modo normal, así que no era necesario empezar desde cero. Con una serie de funciones nuevas y algunos retoques tanto al código original como a la hoja de estilos, se pudo implementar este modo en cuestión de un par de días.

- **Registro de estadísticas y tabla de clasificación:** La propuesta de proyecto original no contemplaba la necesidad de registrar las estadísticas de los usuarios más allá de los elementos que hubieran descubierto. La sugerencia por parte del tutor de añadir algunas características que necesitaran consultar la base de datos para funcionar fue lo que hizo pensar al autor que sería buena idea añadir estadísticas visibles en la pantalla del usuario. Fue necesario modificar los modelos de Django existentes para acomodar nuevos datos que permitieran el correcto seguimiento de las estadísticas así como su actualización automática.

  Una vez realizadas las migraciones pertinentes de la base de datos, se tuvo que agregar a la vista principal del proyecto la lógica necesaria para hacer las consultas a la base de datos e incrementar el valor de los campos de estadísticas a medida que el jugador descubría nuevas combinaciones o elementos.

  Con las estadísticas ya funcionando y mostrándose por pantalla, se decidió que se podría expandir este sistema incluyendo un botón que mostrase una tabla de clasificación con los diez jugadores cuyas estadísticas fueran las más elevadas de todas las registradas en la base de datos. Dado que ya se habían hecho las modificaciones necesarias para implementar las estadísticas, añadir el marcador fue cuestión de escribir un bucle que recorriera la base de datos en busca de los diez usuarios con, por ejemplo, más elementos descubiertos para guardar ese valor en una variable y añadirla a la hora de renderizar la vista de Django, permitiendo así acceder a estos valores desde el código HTML.

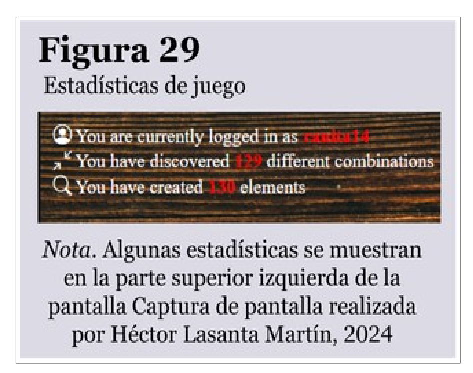


Figura 29
Estadísticas de juego

Nota. Algunas estadísticas se muestran en la parte superior izquierda de la pantalla Captura de pantalla realizada por Héctor Lasanta Martín, 2024

- **Estilos CSS dependientes de las estadísticas del usuario:** Aprovechando una vez más el trabajo realizado a la hora de implementar las estadísticas, al autor se le ocurrió alterar el estilo de algunos elementos dependiendo de si el usuario actual fue quién los descubrió o no. Esto se hace mediante un campo de valor "Muchos a muchos" presente en la tabla que guarda la información de los elementos de la base de datos, explicado anteriormente en más detalle dentro del apartado "Modelos" dentro de "Realización del proyecto".

Cuando la aplicación detecta que el usuario actual fue quién descubrió un elemento, modifica su estilo para mostrar una sombra de color magenta al pasar por encima de ellos con el cursor en vez de la sombra gris que suele acompañar a los elementos descubiertos por otros usuarios.

Además, al generar un nuevo elemento, la animación que rodea el *pop-up* que se muestra por pantalla también ve su color modificado para que se vea magenta en lugar de rojo.

Estos cambios podrían parecer insignificantes, pero cada pequeño detalle puede suponer una gran diferencia.

Figura 30
Estilos dependientes del usuario

Nota. Algunos estilos, como el sombreado morado, solo se muestran si el usuario actual descubrió el elemento. Captura de pantalla realizada por Héctor Lasanta Martín, 2024

- **Tutorial:** Nuevamente, esta idea se originó en una sugerencia del tutor. Añadir un tutorial en condiciones implicaba hacer uso de *cookies* para evitar que se mostrara cada vez que el usuario accediera a la página, pero asegurándose de que apareciera al menos la primera vez de forma automática para que los nuevos usuarios no lo pasaran por alto sin querer.

  Trabajar con *cookies* desde Django supuso cierto trabajo de investigación, pues la única experiencia que tenían el autor y sus compañeros de clase sobre el tema (gracias al módulo de implantación de aplicaciones web) eran nociones teóricas básicas sobre el funcionamiento de las *cookies* y su manejo mediante PHP.

  No resultó complicado extrapolar dichos conocimientos al desarrollo con Django. De hecho, fue lo suficientemente sencillo como para que se decidiera implementar *cookies* en otros lugares de la aplicación, como la página de inicio o el modo desafío.

  Para que el tutorial consiguiera transmitir la información de manera correcta, se prepararon unos cuantos GIFs explicativos que ayudan bastante a entender los conceptos explicados en el tutorial.

Propuestas de mejora

Es del interés del autor empezar esta sección expresando su satisfacción con el trabajo realizado y el correspondiente resultado. Cree que cumple con todo lo especificado en la propuesta de proyecto inicial y añade mucho más. Sin embargo, el proyecto queda lejos de ser perfecto, y la perfección nunca fue algo que se esperase conseguir. Es por esto que a continuación se listan una serie de cambios y mejoras que podrían acercar el proyecto a esa inalcanzable meta:

- **Mejoras de la interfaz de usuario:** Aunque la interfaz cumple con su objetivo y permite a los usuarios jugar a AIquimia sin demasiadas complicaciones, se podrían encontrar formas de distribuir mejor los botones y de cambiar aún más la forma en la que la interfaz se ve en distintos tamaños de pantalla para que la experiencia de juego sea lo mejor posible independientemente del dispositivo utilizado por el usuario para acceder a la aplicación.

- **Mejoras visuales:** Resulta evidente que el autor no es ningún diseñador gráfico y los estilos CSS que ha creado manualmente para la aplicación web son bastante pobres en comparación con aquellos desarrollados por profesionales que entienden del tema. Podría parecer algo no muy importante, especialmente teniendo en cuenta que Little Alchemy, el juego original en el que se inspira este proyecto, no es mundialmente conocido por su asombroso apartado gráfico. Aún así se mantiene la creencia firme por parte del autor de que una pequeña mejora visual de la página web puede incrementar en gran medida la opinión que sus usuarios forman de la misma.

- **Más opciones:** Añadir un menú de configuración que permitiera a los usuarios mejorar su experiencia mediante temas personalizados, distintos tamaños de letra y fuentes, selección de idioma, selección de modo de pantalla (claro / oscuro) etc. podría ser una mejora relativamente sencilla de implementar que aportaría bastante al proyecto.

- **Objetivos mejor definidos:** En su estado actual, el modo principal de AIquimia es un *sandbox* que permite jugar sin tener un objetivo claro. Para intentar solucionar esto, se implementó el modo desafío, que como se explicó anteriormente tiene un objetivo concreto que alcanzar. A pesar de que ese modo mejora la situación, todavía se podría seguir refinando la idea hasta desarrollar algún tipo de sistema de progresión, aunque fuera meramente cosmético, para incentivar a los usuarios a que jugasen más.

- ***Prompts* más precisos:** A la hora de hacer consultas a la IA, fueron utilizados unos *prompts* que funcionan como se espera, pero que no son ideales para la comunicación entre un ser humano y una IA. Adaptar estos *prompts* para que Llama2 pueda interpretarlos mejor podría reducir el número de errores y mejorar la calidad de las respuestas.

- **Implementación de un servidor WSGI:** Ya se mencionó en el apartado dedicado a Nginx, pero el uso de una WSGI para desplegar una aplicación de Django no solo es el estándar para aplicaciones creadas con Python, sino que sería prácticamente obligatorio si no se tratara de un proyecto académico, ya que mejoraría considerablemente la seguridad y estabilidad de la aplicación.

  Algunas de las opciones existentes con las que se podría hacer esto son Gunicorn, uWSGI o Apache (utilizando el módulo mod_wsgi y adaptando los ficheros de configuración para poder trabajar con la base de datos de usuarios de Django).

- **Conversión del proyecto en una aplicación web progresiva (PWA):** Una PWA es un software de aplicación que se reproduce directamente en el navegador en lugar de necesitar ser instalada en el teléfono. Esto permitiría convertir la aplicación web en una aplicación de Android de manera relativamente sencilla, ampliando el número de usuarios que podrían disfrutar de la misma sin necesidad de desarrollar una aplicación móvil desde cero.

- **Mejor uso de funciones:** Durante el desarrollo ha aparecido la necesidad de reutilizar código de distintos archivos, pero al no haberse hecho un uso correcto de las funciones, se ha terminado con bastante código duplicado. Debido al avanzado estado del proyecto, se decidió dejarlo así, pero si se tuviera en mente seguir trabajando en esta aplicación web, una de las primeras cosas que deberían hacerse sería sin duda limpiar el código actual, de forma que fuera mucho más legible y manejable.

Ideas descartadas

Prácticamente todo lo establecido en el anteproyecto ha llegado a manifestarse de una forma u otra en el resultado final. No obstante, durante el proceso de desarrollo hubo una serie de ideas que parecían prometedoras, pero por un motivo o por otro, tuvieron que ser descartadas. A continuación se hablará un poco sobre dichas ideas, explicando por qué se pensó que serían buenas y por supuesto, por qué se acabaron considerando inviables:

- **Generación de imágenes con IA:** Una de las primeras ideas que se sugirió por parte del profesorado del centro fue generar las imágenes de los elementos con inteligencia artificial para que la página fuera más impresionante y vistosa a nivel visual. Desde un principio, el autor no estaba muy convencido de que esto fuera a ser una buena idea, principalmente porque las herramientas de generación de imágenes actuales suelen producir resultados bastante inconsistentes a nivel visual, lo que produciría un popurrí de estilos visuales excesivamente heterogéneo para su gusto (Tampoco es un problema muy serio, pero igualmente se consideró que era una mejor opción que todas las imágenes provinieran de un conjunto ya definido con un estilo homogéneo, como es el caso del pack de emoticonos utilizado).



Figura 31
Icono que representa el elemento "Army"

Nota. Las imágenes generadas con IA pueden quedar bien, pero no suelen mantener un estilo consistente. Imágenes generadas por Microsoft Copilot, 2024

  Esto ya era suficiente como para que se descartara la idea, pero el verdadero problema que impide implementar esta tecnología es que el tiempo que la IA tarda en generar una imagen es demasiado elevado, especialmente para una aplicación que necesita realizar varias consultas por minuto. Aumentar en gran medida el tiempo necesario para procesar las solicitudes a cambio de unas imágenes más variadas y atractivas no parecía merecer la pena, así que finalmente se utilizó una biblioteca de imágenes gratuitas y se le pidió pedí al LLM que asociara los elementos generados al nombre de imagen que le parecía más apropiado.

- **Uso de un LLM local:** Inicialmente se planteó el uso de un modelo de lenguaje instalado de manera local en una máquina virtual, llegando incluso a solicitarse y recibirse el permiso por parte de Meta para descargar su modelo gratuito de Llama2 disponible para pruebas y experimentación. El problema principal era el elevado consumo de recursos que un modelo de estas características requeriría. Una instancia EC2 que pudiera trabajar localmente con el LLM habría resultado en un aumento desproporcionado de los costes de operación de AWS, y como solo disponía de aproximadamente 80$, el autor no quiso arriesgarse a quedarse sin dinero.

El autor considera necesario mencionar que contaba con la ayuda del tutor de proyecto para implementar todo lo relacionado con el LLM al proyecto, así que la ausencia de tutor durante las primeras etapas de desarrollo hizo que acabara optando por usar una API para realizar las consultas necesarias.

- **Creación de una instancia Amazon S3 Bucket para copias de seguridad:** Cuando el autor relizó el curso de AWS Cloud Foundations para el módulo de Servicios de Red e Internet, descubrió muchos de los servicios ofrecidos por AWS. Tras analizarlos llegó a la conclusión de que podría incluir dos de ellos en el proyecto para aumentar su complejidad y aprender a usar nuevas tecnologías relacionadas con la administración de sistemas. Estos servicios eran RDS y S3 Bucket (Un servicio de almacenamiento masivo de datos y ficheros). El primero pudo integrarlo al proyecto sin problemas, como ya se detalló en su respectivo apartado. Sin embargo, hacer funcionar el S3 junto con la instancia RDS demostró ser una tarea imposible porque la cuenta de AWS proporcionada por el instituto carece de los permisos necesarios.

  Crear un S3 Bucket fue sencillo y se hizo bastante rápido. Se intentó subir manualmente archivos y todo funcionaba correctamente. El problema se presentó cuando fue necesario asignarle el S3 Bucket a la instancia RDS para permitir que las copias de seguridad de la base de datos se crearan automáticamente desde la consola de AWS y se movieran al S3 Bucket.

  Hacer esto requería de un rol específico que había que asignar a la instancia RDS para poder hacer cambios sobre el S3. Para ello, era necesario crear una política que tuviera asignados unos determinados permisos desde el panel de IAM (Identity and Access Management). Hacer esto tampoco llevó mucho tiempo, y en principio parecía que todo iba bien.

  El último paso era asignar la nueva política a un rol de IAM. Este es el paso que impidió usar el S3 Bucket; la cuenta de AWS del instituto no tiene permiso para crear o modificar los roles de IAM. Consecuentemente, no era posible automatizar el traslado de las copias de seguridad de la base de datos al S3 Bucket.

  Ya que la idea de guardar en una máquina separada del servidor y de la instancia RDS las copias de seguridad de la base de datos era buena, se tomó la decisión de reemplazar el S3 por una segunda instancia EC2 que en un principio no estaba planeada y usarla exclusivamente para guardar las copias de seguridad y los *logs*, que habría sido la finalidad del S3.

- **Migración del proyecto a una cuenta personal de AWS:** Ante los problemas ocasionados por la falta de permisos, surgió la idea de crear una cuenta personal de AWS en la capa gratuita para así poder utilizar el S3 Bucket. Esta opción habría obligado a migrar el proyecto en su totalidad a las nuevas máquinas virtuales, lo que habría frenado ligeramente el desarrollo del proyecto. No obstante, la complicación que acabó enviando a esta idea al cubo de basura fue el error de verificación de

tarjeta bancaria que AWS mostraba tras la creación de la cuenta. Tras intentarlo por activa y por pasiva, se descartó la idea para evitar malgastar más tiempo de trabajo.

- **VPN:** Otra de las ideas que surgieron mientras se buscaban formas de expandir el proyecto incluyendo funciones relacionadas con lo estudiado fue la implementación de una VPN enrutada que conectara el portátil del autor con la instancia EC2 que contiene el servidor principal. El propósito de esta VPN sería mejorar la seguridad del proyecto, configurando los grupos de seguridad de AWS para que solo aceptaran conexiones SSH desde direcciones IP de la VPN. De esta forma, podría asegurarse que nadie más pudiera acceder al servidor.

  El problema con esta idea que podría parecer prometedora en un principio radica en el hecho de que aunque seguramente funcionaría y lograría el objetivo propuesto, la medida es totalmente innecesaria, ya que se podría simplemente configurar el grupo de seguridad de AWS para que solo permita conectarse a la dirección IP del autor, lo que consigue exactamente el resultado deseado sin necesidad de implementar la VPN.

  Agregar características innecesarias al proyecto solo para aumentar su complejidad cuando estas no ofrecen nada de valor parecía una decisión estúpida, así que se acabó dedicando el tiempo sobrante a refinar el modo desafío para asegurarse de que quedase libre de *bugs*.

  Uso de iptables con NAT: Cuando se lanzó la instancia EC2 dedicada a almacenar las copias de seguridad y *logs*, el autor pensó en aplicar los conceptos aprendidos durante la práctica de clase "iptables con NAT" para que la nueva máquina pudiera acceder a internet a través de la primera.

  Se configuraron todas las reglas tanto de *prerouting* como de *postrouting* sin ningún problema, pero para que funcionara se necesitaba establecer en la segunda instancia EC2 como puerta de enlace predeterminada la dirección IP del servidor principal.

  Esto habría sido muy sencillo en máquinas virtuales como las que se habían usado a lo largo del curso, ya que en ellas se podían configurar manualmente los ajustes de red desde los propios archivos del equipo. El problema es que la instancias EC2 de AWS no se pueden configurar directamente mediante estos ficheros, o al menos, no se ha aprendido a hacerlo durante el curso.

  El objetivo era evitar asociar una IP pública a la segunda instancia EC2, pero la única alternativa conocida por el autor para lograr esto implicaba la creación de un *gateway* con NAT desde AWS y la posterior asignación de una IP pública a este. Evidentemente, esta solución resultaría más costosa que simplemente asociar de manera temporal una IP elástica a la instancia EC2, así que se decidió descartar la idea por completo.

Conclusiones

Una vez finalizado el proyecto llega el momento de echar un vistazo hacia el camino recorrido y analizar el trabajo realizado.

El proyecto ha cumplido con todo lo planteado en la propuesta original, manteniéndose prácticamente cada detalle mencionado en el anteproyecto. Además de esto, se han incluido un gran número de mejoras y funciones que han permitido al proyecto alcanzar el nivel de complejidad y calidad que el autor cree que se exige para conseguir aprobar el módulo.

Dicho esto, y a pesar de la estrecha relación entre lo estudiado durante el curso y lo puesto en práctica durante las 10 semanas de desarrollo, el proyecto como concepto sigue pareciendo a primera vista un proyecto de desarrollo más que de administración, independientemente de que tras analizarlo en detalle se puedan observar todos esos añadidos que lo convierten en un proyecto apropiado para el módulo.

Si fuera posible volver al pasado y cambiar el proyecto elegido, muy probablemente se optaría por hacerlo. No porque se considere que este proyecto no cumple los estándares que se esperan de él, sino porque las primeras impresiones son muy importantes, y este proyecto, a nivel conceptual, no grita ASIR tanto como otros posibles proyectos.

En este sentido, el autor considera que la elección de un proyecto así, cuyo objetivo era poner en práctica todo lo aprendido en mayor o menor medida a la par que se adquirían nuevos conocimientos fue un error por su parte.

No obstante, y como ya se ha dicho en varias ocasiones, objetivamente el contenido del proyecto se adecúa a lo estudiado y por tanto, es un proyecto perfectamente válido y justificado. Definitivamente queda lejos de ser el proyecto ideal, pero teniendo en cuenta lo explicado y dadas las circunstancias que rodean al curso (principalmente la ausencia de varios profesores a lo largo de los dos años), el autor considera que el proyecto es lo suficientemente bueno como para cumplir con los objetivos del módulo y por lo tanto, considera que ha hecho un buen trabajo (a pesar de que por supuesto siempre podría haber sido mejor).

Recursos utilizados para la realización del proyecto

A modo de *webgrafía*, se incluye a continuación una lista con todos los recursos utilizados durante el proceso de desarrollo de este proyecto, ordenados en primer lugar por orden alfabético de las páginas web / recursos principales y a continuación en el orden en el que fueron consultados los apartados concretos.

Entre los enlaces se encuentran tanto los medios de los que he obtenido información / código; como aquellos de los que he tomado imágenes, siempre con el permiso de su propietario y respetando las licencias pertinentes.

- Ask LibreOffice
  - How do I begin page numbering from the second page
- AWS
  - Connecting to a DB instance running the MariaDB database engine
  - Connect to the internet using an internet gateway
- Bootstrap
- Canva
  - Free GIF Maker
- certbot
  - Nginx on Debian 10
- ChatGPT
- CloudDevs
  - How to set up HTTPS (SSL) for a Django website?
- CODEDEC
  - How to create login and registration in Django
  - User Login and Logout in Django
- Copilot
  - Configuración de logging
  - Generación de imágenes
- dbDiagram.io
- DigitalOcean
  - How To Use MySQL or MariaDB with your Django Application on Ubuntu 14.04
  - How To Configure Nginx to Use Custom Error Pages
- Documentación oficial de Django
  - Models
  - Using the Django authentication system
  - Settings
  - Many-to-many relationships
  - Making queries
  - Complex lookups with Q objects
- Figma

    - Free Emoji Pack
- Flaticon
    - Computer icon
    - Database icon
    - Router icon
    - Internet icon
- Freecodecamp
    - Entornos virtuales de Python explicados con ejemplos
- FreeDNS
- geeksforgeeks
    - Python | datetime.timedelta() function
- GitHub
    - Replicate Python client
- Hostalia
- IONOS
    - CSRF: explicación del ataque Cross Site Request Forgery
- iStock
    - Textura de fondo de madera marrón vintage con nudos y agujeros de uñas - Foto de stock
    - Papel vintage con espacio para texto o imagen - Foto de stock
- JavaScript Tutorial
    - JavaScript Anonymous Functions
- JSDELIVR
- Let's Encrypt
    - Comenzando
- Linuxize
    - How to Set or Change Timezone on Debian 10
- MAKEUSEOF
    - How to Create a Custom 404 Error Page in Django
- MDN Web Docs
    - Modelo de Objetos del Documento (DOM)
    - Document: DOMContentLoaded event
    - Element: evento keyup
- pixabay
    - Cosmos Milky Way Night
- PythonTUTORIAL
    - Django ORM
    - Django LoginView
    - Django Cookies
- Real Python

  - Python's F-String for String Interpolation and Formatting
  - Python's Counter: The Pythonic Way to Count Objects
- Reddit - Django
- Replicate
  - Run a model from Python
  - meta / llama-2-7b-chat
    - Streaming
  - A guide to prompting Llama 2
- Snapcraft
  - Installing snap on Debian
- Stack Overflow
  - What does "function anonymous" mean in Javascript?
  - How can I center text (horizontally and vertically) inside a div block?
  - Django - makemigrations - No changes detected
  - CSRF Token missing or incorrect
  - Python pip install mysqlclient
  - ¿Que función tiene el "(e)" en los códigos Js?
  - How to specify the login_required redirect url in django?
  - Using LoginView and LogoutView with custom templatesUsing LoginView and LogoutView with custom templates
  - Adding class to sweet alert
  - How to force desktop view on mobile devices?
  - How can I move EC2 instances to a different subnet?
  - Custom Bad Gateway Page with Nginx
  - Why does DEBUG=False setting make my django Static Files Access fail?
- SweetAlert
  - Installation guide
- TECHSTACKER
  - How to put elements at the bottom of its container with CSS
- W3Schools - CSS
  - CSS Box Sizing
  - CSS outline Property
  - CSS Animations
- W3Schools - JavaScript
  - JavaScript For Of
  - HTML DOM Element cloneNode()
- Wikipedia
  - Nginx
  - Django (Framework)
  - Advanced Packaging Tool

- Pip (administrador de paquetes)
- Youtube - Hedy Graphics
  - FREE EMOJIS PNG I FREE DOWNLOAD
- Youtube - THE PROTON GUY
  - Insert Data to Database from HTML and CSS Form in Django | Django Database Form
- Youtube - UskoKruM2010
  - Django y JavaScript: Peticiones AJAX con Django (Carga dinámica de datos) ✅
- Youtube - Web Dev Simplified
  - How To Build Sortable Drag & Drop With Vanilla Javascript

Índice de figuras

Agradecimientos especiales

Yo, Héctor Lasanta Martín, autor de este documento y del proyecto en sí, quiero aprovechar esta pequeña sección al final del mismo para dirigirme en primera persona, y dejando las formalidades a un lado, a todos aquellos que han hecho que este proceso sea posible.

En primer lugar, querría darle las gracias a todo el equipo formativo del I.E.S. Francisco Romero Vargas con el que he tenido la suerte de compartir estos dos años de mi vida, que suena poco, pero es un 10% de los años que llevo vivo, así que diría que es una fracción importante de dicho tiempo. Dicen que Dios no castiga dos veces, pero claramente eso es mentira, ya que habéis tenido que aguantarnos y además echarle un vistazo a estos proyectos. Sé que es vuestro trabajo, pero alguien tiene que hacerlo y creo que se merece su agradecimiento.

En segundo lugar me gustaría darle las gracias al maravilloso equipo de betatesters que han ayudado a encontrar y corregir *bugs*. Dicho así queda muy profesional, pero hablo de ranita14 (mi madre, a la que por algún motivo que escapa mi comprensión le entretenía el juego más de lo esperado) quien gobierna las tablas de clasificación con sus más de cien elementos creados (más otros cientos que se perdieron durante el desarrollo), y de mi amigo y magnífico compañero de prácticas Jaime, al que es posible que conozcan por su apodo p0yamixta y que me permitió poner a prueba las magníficas herramientas de administración de Django gracias a las cuales su nombre no será recordado por la historia.

En tercer lugar creo que es importante agradecer al resto de mi familia y amigos, que con sus comidas familiares, partidos de fútbol, conciertos en Fortnite Festival y problemas informáticos cuya solución es más sencilla incluso que apagar y volverlo a encender, me han ayudado a procrastinar y a dejar para mañana lo que debería hacer hoy. A veces es necesario tener momentos así, y agradezco que decidan compartirlos a mi lado.

En cuarto lugar, me doy las gracias a mí mismo, porque vamos a dejar las cosas claras, todas las personas ya mencionadas han ayudado mucho y se merecen su reconocimiento, pero… ¿Yo también, no? Puede que el proyecto no sea la gran cosa y nunca pretendió serlo, pero yo estoy orgulloso del trabajo que ha habido detrás del mismo independientemente de lo que se esperase de mí. He sido constante, algo que a pesar de mis nueves y dieces sin valor alguno nunca había conseguido ser. Siempre me había resultado imposible dedicarle más de dos días a cualquier cosa, y por una vez… Lo he hecho. Así que gracias, yo, hiciste un trabajo aceptable, pero no te vengas arriba todavía.

Para terminar los agradecimientos, y por tanto el proyecto, quería darte las gracias a ti, persona que está leyendo esto, porque será un topicazo mayúsculo, pero te lo agradezco más de lo que mis palabras pueden transmitir. Me gustan los videojuegos de gestión de recursos desde que era pequeño, es por esto que entiendo que el recurso más valioso que tenemos los seres humanos es el tiempo. Pensar que alguien esté empleando parte del suyo leyendo esto, dándole sentido al tiempo que yo mismo he invertido en escribirlo, es todo un honor. Este escrito no es merecedor del tiempo de nadie, así que independientemente de si ya te lo he agradecido o no…

GRACIAS